Construcción del lenguaje ñ

LAURA DEL PINO DÍAZ Y AITOR DEL PINO SAAVEDRA HERNÁNDEZ – GRADO EN INGENIERÍA INFORMÁTICA – MENCIÓN EN COMPUTACIÓN

Procesadores de Lenguaje

ÍNDICE

|  |  |
| --- | --- |
| Introducción | 2 |
| Definición del lenguaje Ñ | **3** |
|  |  |
|  |  |

INTRODUCCIÓN

En este documento definiremos la creación de un nuevo lenguaje de programación denominado Ñ. Este lenguaje estará escrito en español y tendrá una base la cual estará formada por funcionalidades de diferentes lenguajes de programación, entre los que destacan MATLAB, Java y Korn Shell.

Por ello en la siguiente sección modelaremos como va hacer nuestro lenguaje de programación con el objetivo de poder crear nuestro compilador en C.

DEFINICIÓN DEL LENGUAJE Ñ

TIPOS DE DATOS

En esta sección definiremos el modelo de nuestro lenguaje. El lenguaje Ñ estará formado por tres tipos de datos:

* Reales. Este tipo lo utilizaremos para codificar cualquier número.
* Caracteres. Este tipo nos va a permitir poder representar lo caracteres del código ASCII.
* Vectores. Este tipo lo utilizaremos para almacenar un conjunto de caracteres o de números en un espacio de memoria contigua y que podremos referenciar mediante un nombre.

|  |  |
| --- | --- |
| Tipos. | Ejemplos. |
| Reales. | 5, -1 , 3.05 |
| Caracteres. | A,b,5,-,\* |
| Vectores. | [1,-2,3.05,-50.10],[H,o,l,a] |

Para declarar una variable de tipo real se puede utilizar las siguientes gramáticas:

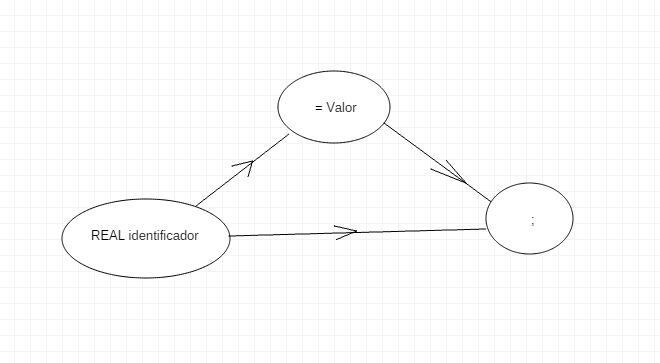


Ilustración : declaración de una variable real

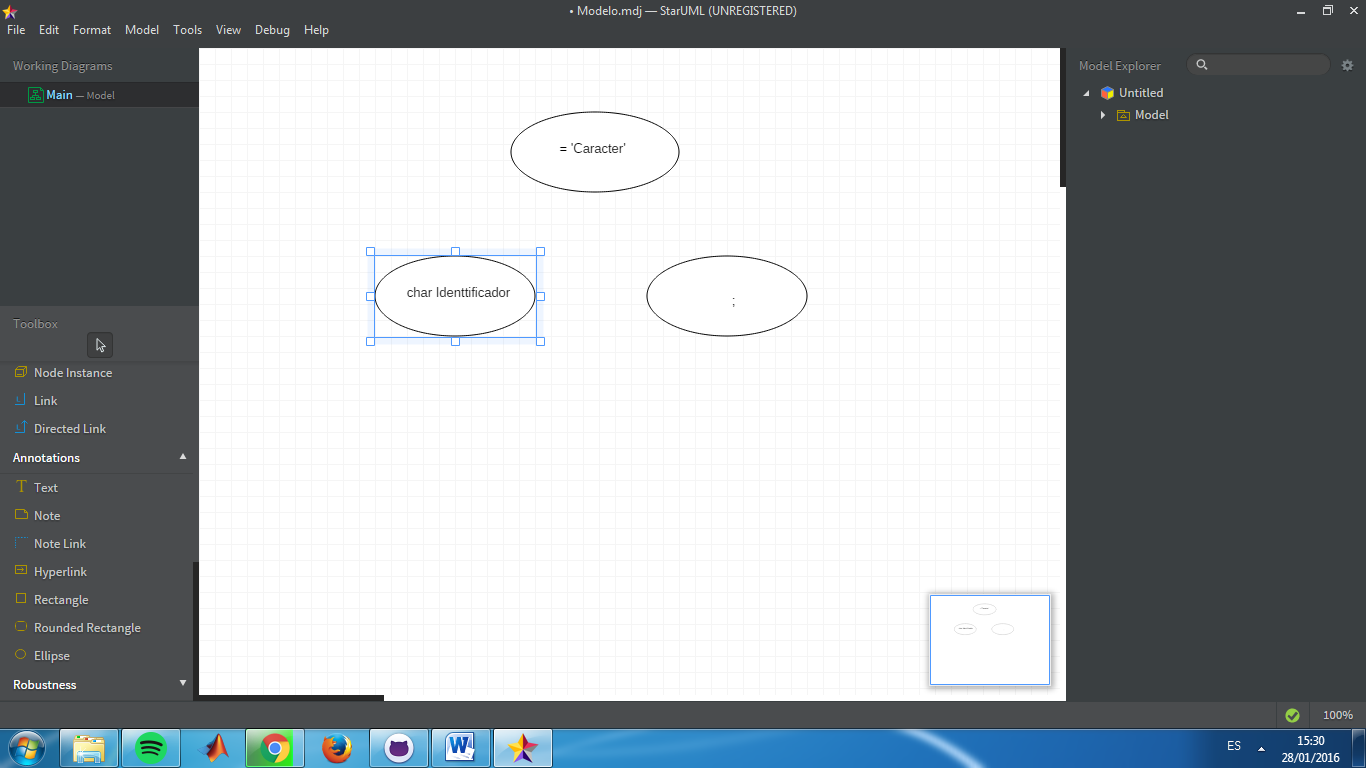


Ilustración 2: declaración de una variable de carácter.

En este lenguaje dispondremos de un conjunto de operadores bastante grandes, lo que nos permitirá realizar prácticamente cualquier operación de manera directa. Dispondremos de operadores aritméticos, relacionales y lógicos.

OPERADORES ARITMÉTICOS

Los operadores aritméticos los utilizaremos para realizar operaciones de tipo numérico por tanto se utilizarán el tipo real o un vector con un conjunto de números.

|  |  |  |
| --- | --- | --- |
| Operador | Descripción | Ejemplo |
| + | Suma | 5 + 2 = 7 |
| .+ | Suma punto | [1,2] + [2,3] = [3,5] |
| - | Resta | 5-3 = 2 |
| .- | Resta punto | [1,3.5,2] – [1,0,-3] = [0,3.5,-1] |
| \* | Producto | 3 \* 5 = 15 |
| .\* | Producto punto | [1,2,5]\*[1,2,3] = [1,4,15] |
| / | División | 20/5 = 4 |
| ./ | División punto | [10,15,30] ./ [2,3,10] = [5,5,3] |
| % | Módulo | 4%2 = 0 |
| .% | Modulo punto | [4,3,1] .% [4,3,1] = [0,0,0] |
| ++ | Incremento | 5++ = 6 |
| -- | Decremento | 5-- = 4 |

Además podremos realizar sumas, restas, multiplicaciones y divisiones de un valor por un vector, de manera que el resultado será realizar la operación de ese valor por todos los valores que componen el vector. Por ejemplo 5 + [1,2,3] = [6,7,8].

OPERADORES RELACIONALES

También tendremos operadores relaciones los cuales utilizaremos para realizar una comparación numérico o de caracteres entre dos operandos. Exactamente los caracteres utilizarán únicamente los operadores de igual o distinto. El resultado de la evaluación de cualquier operador relacional será 1 o 0, 1 si se cumple esa comparación y 0 sino se cumple.

|  |  |  |
| --- | --- | --- |
| Operador | Descripción | Ejemplo/Resultado |
| == | Igual | 5 == 5 /1 |
| ¡= | Distinto | a != b /0 |
| > | Mayor | 5 > 0 /1 |
| < | Menor | 5 < 4 /0 |
| >= | Mayor o igual | 5 >= 2 / 1 |
| <= | Menor o igual | 10 <= 5 / 0 |

OPERADORES LÓGICOS

Los operadores lógicos se aplican en operaciones relacionales con el objetivo de poder evaluar varias condiciones simultáneamente. En el caso de que se cumpla la condición el resultado será un 1, en caso contrario será un 0.

|  |  |  |
| --- | --- | --- |
| Operador | Descripción | Ejemplo/Resultado |
| && | AND | (5 == 5 && A == a) / 0 |
| || | OR | (2 !=1 || 10.5 >= 2) / 1 |
| ! | NOT | !( 2 == 1)/1 |

SENTENCIAS CONDICIONALES

En el lenguaje Ñ dispondremos de dos sentencias condicionales, la sentencia if-else para condiciones simples y la sentencia switch para condiciones múltiples.

La sentencia if-else nos permite decidir entre dos posibles opciones excluyentes. La expresión que acompaña al if debe producir al ser evaluada un 0 o 1 dependiendo de si se cumple o no. En el caso que se cumpla, se ejecutarán el conjunto de sentencias que se encuentra dentro del if. En caso contrario se ejecutarán las sentencias que se encuentran dentro del else si está ya que dicho bloque es opcional.

La sintaxis es la siguiente y como dijimos anteriormente la parte del else es opcional.

if ( expresión){

sentencia-k

}else{

sentencia-k

}

if (expresión){

sentencia-k;

}else{

sentencia-k;

}

La sentencia switch nos permite decidir entre múltiples opciones excluyentes. El funcionamiento es muy sencillo se evalúa la expresión y en caso de coincidir el valor de la expresión con el valor de una de las ramas case se ejecuta el conjunto de sentencias que sigue hasta el final del switch o hasta encontrarse la sentencia break. Lo normal es que después de cada case o rama haya un break ya que si no se ejecutarían sentencias que no corresponde con dicho case. Además hay un case denominado default el cual es opcional y se ejecuta si no se encuentra coincidencia entre el resultado de la expresión y los cases que se encuentra en el switch.

Existe la restricción debe que debe aparecer primero los case y después el default. Además todas las opciones deben ser diferentes.

K = 2 ... ∞

K = 2 ... ∞

switch (expresion){

case valor1:

sentencias;

break;

case valorK:

sentencias;

break;

default:

sentencias;

}

break;

}

BUCLE WHILE

Este tipo de bucle nos permite ejecutar un conjunto de sentencias repetitivamente mientras se cumpla una determinada condición. Una característica del bucle while es que se ejecuta 0 o N veces ya que si la condición del bucle no se cumple no se entra a ejecutar las sentencias.

Por ello su funcionamiento es muy sencillo, se evalúa la expresión que está en while y si como resultado se obtiene un 1, se ejecutan el conjunto de sentencias que se encuentran en el interior del while. Esto se repetirá hasta que la evaluación de la expresión de un 0, por tanto, se deje de cumplir la condición.

La sintaxis de while es la siguiente:

K = 3... ∞

while (expresión){

sentencia-1;

sentencia-2;

sentencia-K

}

BUCLE FOR

El bucle for se utiliza para ejecutar un conjunto de sentencias un número determinado de veces el cual se fija en el interior del bucle.

La sintaxis del for es la siguiente:

for (exp1;exp2;exp3) {

sentencia-1;

sentencia-2;

sentencia-K;

}

K = 3 ... ∞

Pueden omitirse cualquier de las tres expresiones del bucle for, pero los puntos y coma deben permanecer. Las expresiones exp1 y exp3 son asignaciones mientras exp2 es una expresión condicional. En el caso de que no exista exp2 se considera que la condición es siempre cierta.

La exp1 se utiliza para inicializar la variable que controla el bucle, con exp2 controlamos la permanencia en el bucle y con exp3 realizamos modificaciones sobre la variable que controla el bucle para poder llegar a salir de éste.

Además tendremos un bucle for implícito el cual nos va permitir crear vectores con un conjunto determinados de números. Su sintaxis será la siguiente:

Real [] numeros = exp1:exp2:exp3

La exp1 nos indica en que número se empieza, exp2 el incremento que se va a realizar, y exp3 hasta que número vamos a llegar. Exp2 se puede obviar y en dicho caso el incremento será uno en uno.