PROCESADORES DE LENGUAJES

ESPECIFICACIÓN DEL LENGUAJE A IMPLEMENTAR

Versión 2.0

Estudiante erasmus Fabio Catania

Roberto Díaz

TABLA DE LOS CONTENIDOS

INTRODUCCIÓN

El lenguaje cuyo compilador se implementará a lo largo de las fases de la práctica tiene las características de un lenguaje de programación de alto nivel. Este documento trata del lenguaje de programación recién creado, Vod. Este lenguaje, escrito principalmente en español, posee funcionalidades derivadas de otros lenguajes de programación actual tales como Java.

El lenguaje Vod posee la cualidad “Case Sensitive”, por tanto no es lo mismo escribir una palabra en mayúsculas que en minúsculas. Existen una serie de palabras reservadas, las cuales tienen funciones especiales, y para que puedan desempeñar estas funcionalidades especiales, deben ser escritas tal cual se muestra a continuación. No hace falta si existen espacios libres, tabulaciones o líneas vacías entre los objetos del lenguaje.

Más adelante, haremos una descripción completa del lenguaje.

TIPOS DE VARIABLES

Para almacenar y recuperar los datos con los que los programas van a trabajar hay variables, que son referencias a accesos a memoria. El lenguaje consta de variables tanto numéricas como de ristras. Los tipos primitivos usados son ent para los enteros y cad para las ristras. No existe un tipo para los caracteres individuales, que se tratan como cadenas de caracteres. Cada variable tiene un nombre que debe ser único entre las variables del programa. Las variables que no se declaren dentro de una función tienen ámbito global. Además deben seguir las siguientes reglas:

- no puede ser una palabra reservada del lenguaje;
- puede contener cualquier carácter alfanumérico, pero no puede componerse solo de números.

En la fase de declaración de las variables enteras es necesario escribir la palabra *declar* y especificar su tipo y su nombre. Por ejemplo:

declar ent edad;

La inicialización de un entero es la asignación del valor. Se coloca un signo de igual después del nombre de la variable y luego el valor para asignar. Por ejemplo:

edad = 18;

También se pueden declarar e inicializar en la misma línea. Por ejemplo:

declar ent numero = 1;

La declaración y la inicialización de cadenas se realizan a la vez. Esto significa que una variable de este tipo nunca puede cambiar su valor. Después de la palabra *declar*, del tipo y del nombre de la variable, es necesario poner el signo de igual y el valor escrito entre dos comillas. Por ejemplo:

declar cad título=“El principito”;

Un entero puede ser cero, positivo o negativo. Una ristra puede tener caracteres alfanuméricos en mayúsculas y en minúsculas y los caracteres ‘-’, ‘.’, ‘,’, ‘?’, ‘!’, ‘¿’, ‘¡’, ‘;’, ‘_’, ‘*’ y ‘”. Además, puede contener el carácter especial ‘#’, que la rutina de impresión en la pantalla interpreta como la intención del programador de saltar a nueva línea. Siguen unos ejemplos de posibles valores que se pueden asignar a las variables.

Tipos	**Ejemplos**
ent	-5, 0, 5
cad	“Hola Mundo”, “a#”, “b”, “----“

Siempre que se declara una variable, o cuando se escribe una sentencia sencilla, la línea tiene que finalizar con el símbolo ‘;’. Pueden existir también sentencias vacías. Todas las sentencias se deben considerar sencillas, excepto las estructuras de control ‘mientras’ y ‘si’. La sintaxis de estas estructuras de control se muestra más adelante.

OPERADORES

Operadores Aritméticos

Este lenguaje es muy sencillo y por esto permite solo operaciones entre dos termines, que pueden ser tanto dos variables enteras como dos números. Por eso, tampoco tiene sentido hablar de prioridad entre los operadores a la hora de ejecutarse. Hay que decir que en este contexto se permite rodear el nombre de las variables y las operaciones en sí con paréntesis. El resultado de cada operación tiene que guardarse en una variable. Las operaciones aritméticas posibles son las siguientes:

Operador	**Descripción**	**Ejemplo**
+	Suma	a = 2 + 2;
-	Resta	b = 2 – 4;

*	Multiplicación	c = c * a;
/	División	d = a / a;

Dado que se trabaja solo con números enteros, la operación de división es muy sencilla y no tiene en cuenta del resto. Por eso, por ejemplo, 5/4 será igual a 1.

Operadores para las cadenas

En el caso de las ristras, el único operador que se utiliza es el *conc*, con el objetivo de realizar la concatenación de caracteres. Se trata de una operación binaria entre variables, y su resultado tiene que asignarse a una variable. Además, ya que el valor de las cadenas se puede asignar solo en la fase de inicialización, el modelo de utilización del operador *conc* es el que sigue:

declar cad título= primeraParte conc segundaParte;

Al contrario que en Java, en Vod se pueden concatenar solo caracteres con caracteres, no permitiendo la concatenación entre variables de distinto tipo.

Operadores Relacionales

En este lenguaje los operadores relacionales se utilizan solo entre variables numéricas para realizar una comparación entre ellas. Las comparaciones devuelven un valor numérico que puede ser cero, en el caso que la condición no sea respectada, o uno, al contrario.

Operador	**Descripción**	**Ejemplo**
==	Igual	a == b
!=	Distinto	d != c
>	Mayor	a > e
<	Menor	h < g

ESTRUCTURAS DE CONTROL

En Vod disponemos de dos tipos de sentencias de control; ‘si’, para condiciones simples y ‘mientras’, cuando queremos ejecutar un conjunto de sentencias hasta que se cumpla una condición.

En este lenguaje hablamos de condiciones cuando nos referimos a comparaciones entre variables de tipo entero.

Dentro de una estructura de control no se puede declarar ningún tipo de variable, aunque si se trata de una variable entera, sí se puede modificar su valor.

Sentencia si

Al utilizar la sentencia ‘si’, debemos comparar una operación relacional cuyo resultado sea verdadero. En caso positivo se ejecuta las sentencias incluidas dentro del ‘si’.

Sintaxis:

si (Condición){

Conjunto de sentencias;

}

Sentencia mientras

La sentencia ‘mientras’, nos permite ejecutar una, o un conjunto de sentencias en bucle hasta que se cumpla una cierta condición.

Sintaxis:

mientras (Condición){

Conjunto de sentencias;

}

Las condiciones ‘si’ y ‘mientras’ pueden estar contenidas una dentro de la otra, y también pueden haber varias de un mismo tipo anidadas.

FUNCIONES

Otra de las características del lenguaje Vod, es la posibilidad de escribir funciones, las cuales sirven no solo para dar la posibilidad de crear rutinas recursivas, sino para poder proveer al programador de un conjunto de herramientas para modularizar el código. Cada vez que se crea una función, esta va precedida de la palabra especial *funcion* seguida del nombre de dicha

función. En este lenguaje todas las funciones se asumen como si fueran funciones void que no retornen nada. Las funciones no poseen parámetros, pero pueden comunicar con el exterior mediante las variables globales. Dentro de una función en cualquier lugar de esa se podrán crear variables locales, las cuales son dinámicas por lo que no pueden ser accedidas desde fuera de la función y no pueden tener el mismo nombre de otras variables en el programa. El cuerpo de la función siempre se ejecuta todo hacia el final.

Sintaxis:

- Función

funcion nombre(){

DeclaracionVariableLocal1;

…

DeclaracionVariableLocalM;

Sentencia1;

…

SentenciaN;

}

Declaraciones y sentencias pueden alternarse sin ningún problema.

Para llamar a una función que ya ha sido creada se debe escribir el nombre de la función seguido de 2 paréntesis.

Ejemplo:

Sentencia;

nombreFunción();

Sentencia;

COMENTARIOS

Se permiten los comentarios en cualquier parte del código, siempre que se haga uso de los caracteres especiales “/*” para comenzar los comentarios, y “*/” para finalizar los comentarios. Todo el contenido de los comentarios, aunque fueran instrucciones, pasa desapercibido por el lenguaje. Un ejemplo de uso de comentarios sería:

/ Hello World */*

SALIDA

En Vod no hay rutinas de entrada, solo de salida. Se permite la impresión en la pantalla de variables de tipo ent y cad, pero no de los dos a la vez. Además se pueden imprimir directamente ristras de caracteres sin llegar a tener que inicializar una variable de tipo cad. Se utiliza la palabra reservada *imprimir*, seguida por el nombre de la variable o de una ristra, para imprimir entre dos paréntesis.

Función	**Ejemplo**

imprimir(variable);
 	
imprimir(Ristra);
	imprimir(a);
	
	imprimir(“Hola”);

TESTS

A continuación, se proporciona una batería de tests, que sirven tanto de ejemplo de las distintas estructuras como para validar y detectar errores tanto en los aspectos léxico-sintácticos como semánticos.

TEST I

Este programa es muy sencillo. Al principio se declaran dos variables numéricas enteras n y f. Después se inicializan con dos valores 5 y 1. Una vez que se han inicializado las variables es posible utilizarlas: el programa suma los valores de n y f, imprime el valor de la suma, disminuye el valor de n y chequea si n sigue siendo mayor que cero. Si lo es, se repite, en caso contrario se para.

```
declar ent cero;
cero = 0;
```

```
declar ent uno;
uno = 1;

declar ent n;
declar ent f;
n = 5;
f = 1;
mientras(n>cero){
     f = f + n;
     imprimir(f);
     n = n-uno;
}
```

TEST II

En el segundo programa se declaran e inicializan unas ristras. Además, concatenan todas y se las imprime en la misma línea.

```
declar cad titulo="El principito";
declar cad autor="Antoine de Saint-Exupery";
declar cad frase="El autor de ";
declar cad verbo=" es ";
declar cad punto=".";
declar cad frase1 =frase conc titulo;
declar cad frase2=frase1 conc verbo;
declar cad frase3=frase2 conc autor;
declar cad frase4=frase3 conc punto;
imprimir(frase4);
```

TEST III

Este programa calcula el factorial del número 5 usando el bucle mientras y lo imprime en la pantalla.

```
declar ent factorial;
declar ent numero;
declar ent k;
declar ent f;
declar ent cero;
declar ent uno;
declar cad esp = "#";
uno = 1;
cero = 0;
numero=5;
factorial=1;
k=0;
f = numero - k;
```

```
mientras(f>cero){
      factorial=factorial*f;
      k=k+uno;
      f = numero - k;
}
declar cad frase = "El factorial de 5 es ";
declar cad punto = ". ";
imprimir(frase);
imprimir(factorial);
imprimir(punto);
imprimir(esp);
```

TEST IV

En el cuarto programa se verifica si Mario, que tiene 17, es mayor de edad.

```
declar ent anos;
declar cad nombre = "Mario es mayor de edad.#";
declar cad no = "Mario no es mayor de edad.#";
anos=17;
declar ent eda;
eda = 18;
si(anos>eda){
      imprimir(nombre);
}

si(anos==eda){
      imprimir(nombre);
}

si(anos<eda){
      imprimir(no);
}
```

TEST V

En este programa, como antes, se calcula el factorial de 5. Sin embargo, en este caso se realiza una llamada a función para enseñar cómo funciona.

```
declar ent numero;
numero=5;

funcion calculaFactorial(){
   declar ent k;
   declar ent factorial;

       factorial=1;
       k=0;
   declar ent j;
   j = 0;
   declar ent kk;
```

```
   kk = numero - k;
   declar ent uno;
   uno = 1;

      mientras(kk>j){
           factorial=factorial*kk;
                k=k+uno;
      kk = numero - k;
      }

      declar cad frase = "El factorial de 5 es ";
      declar cad punto = ".#";

      imprimir(frase);
   imprimir(factorial);
   imprimir(punto);
}
calculaFactorial();
```

TEST VI

El sexto programa calcula el valor más alto entre los de la secuencia de Fibonacci correspondiente al número 15. Se puede verificar que el lenguaje permite llamadas recursivas a funciones.

```
declar ent a;
declar ent b;
declar ent c;
declar ent d;
declar ent e;
declar cad esp = "#";
a = 0;
b = 1;
c = 1;
d = 15;
e = 0;
funcion fibo(){
   si(c < d){
      e = b;
      b = a + b;
      a = e;
      d = d - c;
      fibo();
   }
}
fibo();
imprimir(b);
imprimir(esp);
```

TEST VII

En el último test se calcula 34 mediante el auxilio de la función *expo*() y se imprime el resultado en la pantalla.

```
declar ent res;
declar ent expo;
expo = 4;
res = 1;


funcion expo(){
   declar ent uno;
   declar ent cero;
   declar ent base;
   base = 3;
   uno = 1;
   cero = 0;
   si(expo > cero){
      res = base * res;
      expo = expo - uno;
      expo();
   }
}

expo();
imprimir(res);
```

APLICACIÓN

Sigue una pequeña aplicación demostrativa en Vod. Con este código se imprime el valor más grande y el menor valor entre los de diez variables enteras.

```
declar ent i;
declar ent min;
declar ent max;

declar ent uno;
declar ent dos;
declar ent tres;
declar ent cuatro;
declar ent cinco;
declar ent seis;
declar ent siete;
declar ent ocho;
declar ent nueve;
declar ent diez;

uno=3;
dos=5;
```

```
tres=229;
cuatro=9;
cinco=0;
seis=-1;
siete=88;
ocho=46;
nueve=90;
diez=55;

min=uno;
max=uno;

si(dos>max){
      max=dos;
}
si(dos<min){
      min=dos;
}

si(tres>max){
      max=tres;
}
si(tres<min){
      min=tres;
}

si(cuatro>max){
      max=cuatro;
}
si(cuatro<min){
      min=cuatro;
}

si(cinco>max){
      max=cinco;
}
si(cinco<min){
      min=cinco;
}

si(seis>max){
      max=seis;
}
si(seis<min){
      min=seis;
}

si(siete>max){
      max=siete;
}
si(siete<min){
      min=siete;
}

si(ocho>max){
```

```
      max=ocho;
}
si(ocho<min){
      min=ocho;
}

si(nueve>max){
      max=nueve;
}
si(nueve<min){
      min=nueve;
}

declar cad frase = "Entre 3, 5, 229, 9, 0, -1, 88, 46,
90 y 55 el numero mas grande es ";
declar cad segundaparte = " y el mas pequeno es ";
declar cad punto = ".#";
imprimir(frase);
imprimir(max);
imprimir(segundaparte);
imprimir(min);
imprimir(punto);
```

REFERENCIAS

- *http://telepresencial1617.ulpgc.es/cv/ulpgctp17/mod/page/view.php?id=65937*
- *http://telepresencial1617.ulpgc.es/cv/ulpgctp17/pluginfile.php/81007/mod_resource/content/1/tr-intro-Q.pdf*
- *http://telepresencial1617.ulpgc.es/cv/ulpgctp17/pluginfile.php/81009/mod_resource/content/8/ejnano.c*
- *https://docs.oracle.com/javase/tutorial/*