

## **Unidad 4**

### **Manual de Lenguaje de Programación WareLang**

Presentado por:

Acosta Carillo Yvan Fernando  
Ramírez Velázquez Lia Rebeca  
Praiz Cruz Amir Antonio  
Jara Maldonado Javier Valentin

Ingeniería en Sistemas Computacionales, Tecnológico de Tepic

Lenguajes y Autómatas I

M. D. O. H. Sonia Alvarado Mares

12 de abril de 2024

## ÍNDICE

<b>Aplicación.....</b>	<b>3</b>
Comunicación humana.....	4
Prevención y detección de errores.....	5
Usabilidad.....	6
Efectividad.....	6
Compilabilidad.....	7
Eficiencia.....	7
Independencia de la máquina.....	8
Simplicidad.....	8
Uniformidad.....	8
Ortogonalidad.....	8
<b>Diseño Detallado.....</b>	<b>9</b>
Alfabeto.....	9
Tipos de datos.....	9
PUERTO.....	9
VOL.....	9
ENT.....	10
COLOR.....	10
DEC.....	10
BOOL.....	10
CAD.....	10
Grupos de Cadenas.....	10
Operadores relacionales.....	10
Operadores aritméticos y asignación.....	11
Números Enteros.....	12
Números Reales.....	12
Símbolos de agrupación y delimitación.....	12
Operadores lógicos.....	12
Símbolos de puntuación.....	13
Comentarios.....	13
Identificadores.....	13
Palabras Reservadas.....	14
Palabras reservadas para la estructura y flujo del programa.....	14
Palabras reservadas para el funcionamiento del robot.....	14
Microestructura.....	15
Estructura de las expresiones.....	15
Expresiones de concatenación de cadenas.....	15
Delimitacion del programa y métodos.....	15
Definición de un método con DEF.....	16
Métodos del lenguaje.....	16
Método ADELANTE().....	16
Método ATRAS().....	17
Método IZQUIERDA().....	17

Método DERECHA()	17
Método PARAR()	18
Método TOMAR()	18
Método SOLTAR()	18
Método REVISAR()	18
Método CAJA()	19
Método IMPR()	19
Estructuras de datos	19
Matrices	19
Listas enlazadas	20
Estructuras de control	21
Estructuras Secuenciales	21
Estructuras Selectivas	21
Sentencia SI	21
Sentencia MIENTRAS	22
Sentencia REPETIR	22
Estructuras de compilación	23
Análisis léxico	23
Análisis Semántico	23
Generación de código intermedio	23
Optimización del código	24
Generación de código	24
Tablas de símbolos	24
Manejador de errores	24
Estructura del programa	25
Estructuras de la entrada/salida	25
Sensores y Dispositivos	25

## Aplicación.

*WareLang* representa un avance significativo hacia el futuro al proporcionar a las grandes industrias una solución eficiente y efectiva para el traslado y la organización de sus productos. Con su capacidad para automatizar y controlar el comportamiento de robots especializados en la manipulación de mercancías en almacenes, *WareLang* ofrece una herramienta poderosa para mejorar la eficiencia operativa y la productividad.

*WareLang* es un lenguaje de programación especializado en la automatización y control del comportamiento de un robot destinado a la manipulación de productos en un almacén. Este lenguaje está diseñado para gestionar un almacén compuesto por cajas organizadas en una estructura similar a tablas, donde cada caja tiene iguales dimensiones y forma cuadrada. El robot se desplaza a lo largo de vías elevadas sobre estas cajas, permitiéndole levantar y soltar objetos, como se muestra en la Figura 1. Este lenguaje tiene la opción de especificar la dimensión del espacio por el que se desplaza el dispositivo, al desplazarse a lo largo de vías elevadas sobre estas cajas, este puede levantar y soltar objetos.

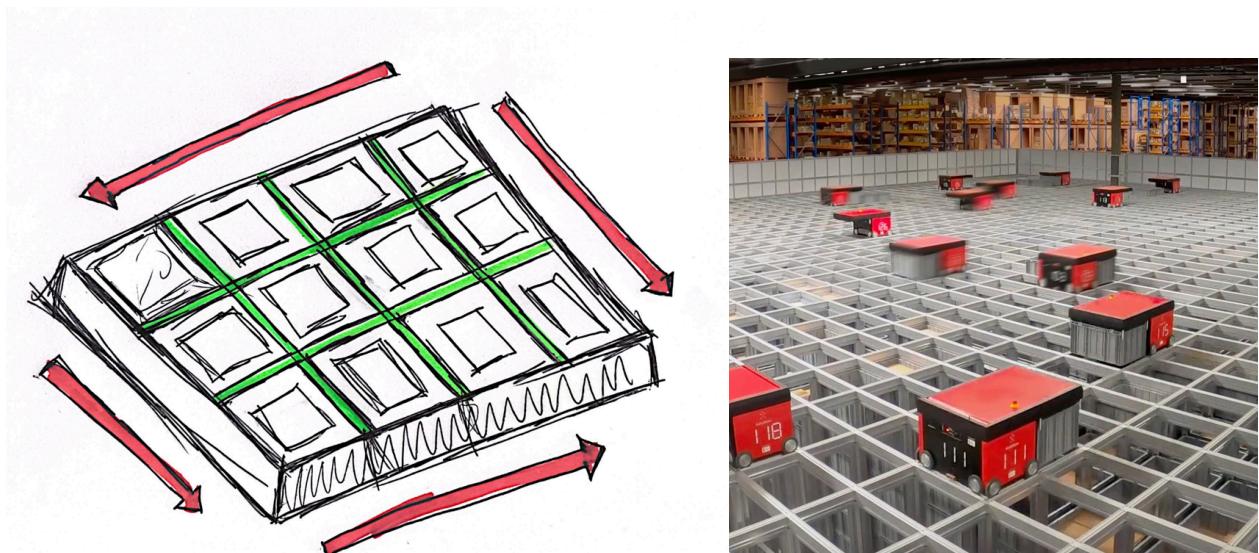


Figura 1. Ejemplificación de la estructura del almacén.

Dicho robot podrá incluso (por medio de métodos posteriormente detallados en su sección) moverse de manera “manual” o automática por el almacén. Al permitir la programación precisa de movimientos y tareas, *WareLang* facilita una gestión más eficiente de los recursos

y una optimización de los procesos de almacenamiento y distribución. Esto se traduce en una reducción de costos y tiempos.

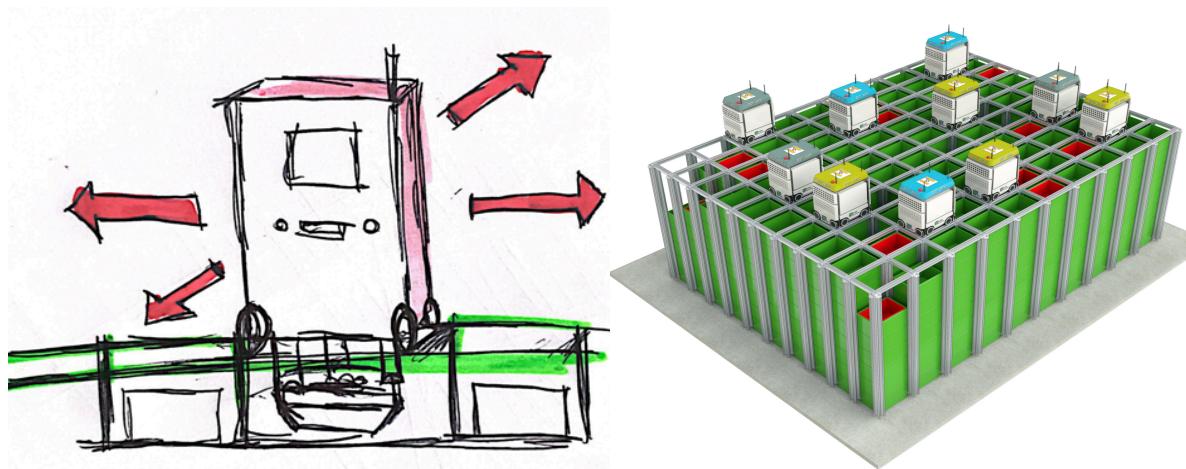


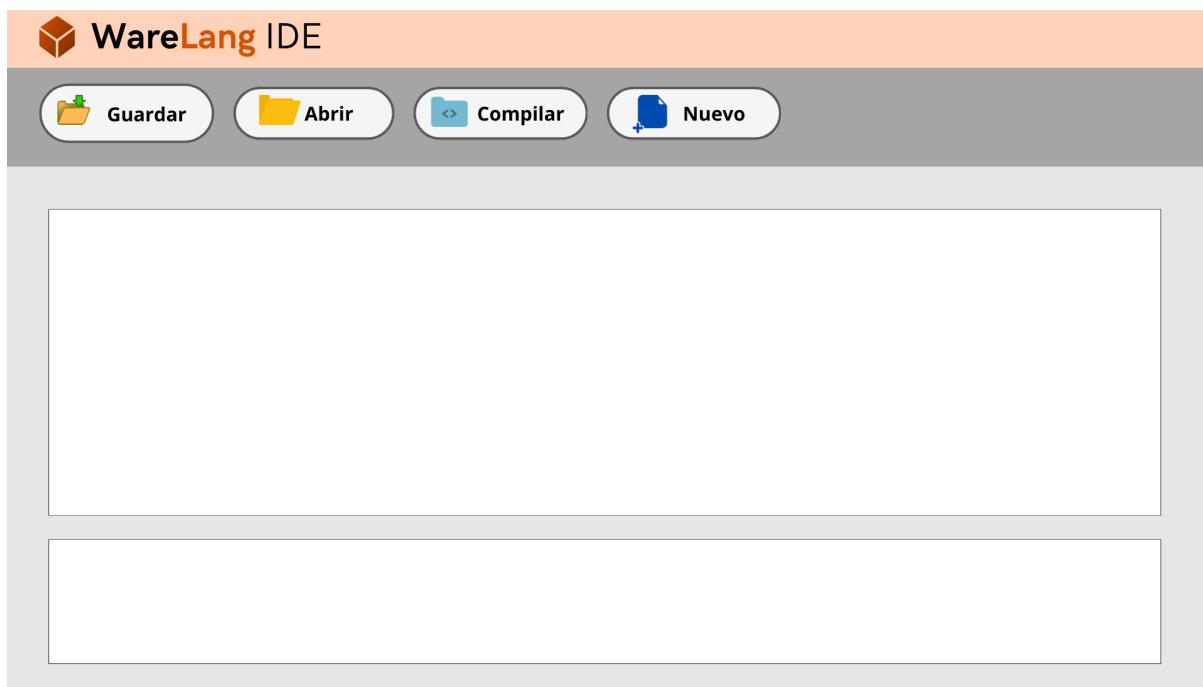
Figura 2. Ejemplificación del movimiento del robot

## Comunicación humana

*WareLang* está equipado con un entorno de desarrollo integrado (IDE) diseñado para mejorar la comunicación entre humanos y máquinas. Este IDE se caracteriza por su interfaz intuitiva y amigable, que incluye una variedad de botones y herramientas diseñadas para facilitar la programación y el control del robot en el entorno del almacén.

Entre las características destacadas del IDE de *WareLang* se encuentran:

- Interfaz visual intuitiva: El IDE presenta una interfaz gráfica fácil de usar que permite a los usuarios interactuar de manera efectiva con el entorno de programación.
- Botones de control: Se proporcionan botones específicos para acciones comunes, como compilar, guardar, nuevo, abrir, etc;
- Herramientas de control de errores: El IDE incluye herramientas que permiten a los usuarios identificar y corregir errores en el código de manera eficiente.



## Prevención y detección de errores

En el lenguaje de programación *WareLang* existen diferentes tipos de errores, como son los siguientes:

- **Errores léxicos:** Estos errores ocurren cuando los lexemas no coinciden con los patrones.
- **Errores de sintaxis:** Estos errores ocurren cuando el código no sigue la sintaxis correcta. Por ejemplo, falta de comillas, paréntesis mal cerrados, o uso incorrecto de palabras clave.
- **Errores semánticos:** Estos errores ocurren cuando las sentencias del código no tienen coherencia o un significado válido, por ejemplo, asignar un tipo de dato incorrecto a una variable, el uso de una variable no declarada, o utilizar una función de manera incorrecta, pero sigue siendo sintácticamente válido.
- **Errores de tiempo de compilación:** Estos errores ocurren durante la fase de compilación del programa y están relacionados con la estructura y organización del código fuente. Por ejemplo, referencias a variables o funciones no definidas, o uso incorrecto de tipos de datos.

El compilador *WareLang* implementa diversos mecanismos para el manejo y prevención de errores, asegurando la integridad del código. Entre las estrategias utilizadas se incluyen:

1. **Documentación del lenguaje:** la documentación del lenguaje es un mecanismo básico y sencillo para la prevención de errores. Por medio de un manual de uso se explica la forma adecuada de codificar, los métodos que existen, los tipos de datos, las palabras reservadas, la sintaxis, etc.
2. **Análisis de sintaxis:** el compilador utiliza un análisis sintáctico sólido para detectar errores de sintaxis en el código.
3. **Mensajes de error:** en el caso del compilador de *WareLang*, los mensajes de error están diseñados para proporcionar información de ayuda que permita al programador comprender el tipo de error cometido y su ubicación en el código fuente.

## Usabilidad

*WareLang* es un lenguaje de programación que destaca por su accesibilidad y facilidad de uso. Su sintaxis sencilla y comprensible se asemeja al lenguaje humano, empleando palabras cortas en español y evitando ambigüedades. Esto no solo reduce la posibilidad de errores, sino que también aumenta la productividad del programador al facilitar la comprensión y escritura del código. Aunque es un lenguaje nuevo, su documentación permite a los programadores entender el uso de este mismo siendo así un lenguaje fácil de aprender.

*WareLang* es un lenguaje de programación diseñado para el uso de un robot organizador de almacén y es capaz de resolver problemas dentro de este contexto. *WareLang* ofrece la capacidad de crear soluciones efectivas a los desafíos que enfrentan los programadores en su ámbito de aplicación.

Cuenta con una herramienta para la depuración y mantenimiento del código llamada *WareLangCompiler*, el cual es un compilador, capaz de ejecutar programas realizados con *WareLang*.

## Efectividad.

Para facilitar tanto la escritura de código como el manejo del mismo durante el proceso de compilación, el lenguaje *WareLang* es tipado, para poder identificar de mejor manera las variables, reduciendo así mismo la cantidad de errores generados. Por otra parte, la sintaxis cuenta con una estructura similar a la de lenguajes populares, haciendo uso de las llaves para agrupar ciertas partes del código, finalizando líneas del mismo con el uso de punto y coma,

generando facilidad de interpretación y análisis para las partes encargadas de ello en el compilador.

Añadido a lo anterior, *WareLang* se ejecuta de manera rápida y segura, pese al tiempo de compilación, los factores mencionados anteriormente tienen peso en un proceso de compilación más seguro y eficiente, gestionando recursos disponibles.

## **Compilabilidad**

En cuanto a la compilación *WareLang* cuenta con *WareLangCompiler* el cual se realiza los procesos necesarios para la traducción del lenguaje de alto nivel, a un lenguaje de bajo nivel interpretable por el procesador del robot. Así mismo, la sintaxis especificada, y el uso de las características planteadas tanto para la estructura del programa como para el uso de variables y su declaración, hacen que el proceso de traducción de alto nivel a bajo nivel sea más eficaz.

## **Eficiencia**

En el lenguaje de programación *WareLang*, se enfatiza la importancia de utilizar eficientemente las estructuras de datos y los bucles para mejorar el rendimiento del programa. Se reconoce que un conocimiento detallado de cada estructura de datos mencionada, así como su comprensión profunda, contribuyen significativamente a optimizar el funcionamiento de la aplicación.

Cuando se implementan bucles en *WareLang*, se promueve el uso de un número mínimo de iteraciones para lograr la eficiencia deseada. Esta práctica se alinea con el objetivo de minimizar la carga computacional y maximizar el rendimiento del programa.

Además, en *WareLang* se reconoce la flexibilidad en la escritura del código, especialmente en la definición de funciones. Se destaca que un mismo conjunto de funcionalidades puede ser implementado de diversas maneras, resultando en código que varía en longitud y complejidad. Esta característica brinda a los desarrolladores la libertad de elegir la forma más adecuada y eficiente de expresar la lógica del programa, según las necesidades específicas del proyecto.

## **Independencia de la máquina**

*WareLang* actualmente se enfrenta a limitaciones de portabilidad, ya que su compilación y ejecución están restringidas a la plataforma Windows. En cuanto a los requerimientos físicos de la máquina se recomienda contar con al menos 2 GB de espacio libre en disco, tener al menos 2 GB de RAM y un procesador de x32, para un rendimiento óptimo.

## **Simplicidad**

*WareLang* es un lenguaje especializado que facilita en gran medida el aprendizaje y dominio del programador hacia este así como el mantenimiento y/o actualización del código. Al no ser de propósito general, la cantidad de estructuras de datos y de control que maneja son menores y dichas estructuras (que se habla a profundidad en su respectiva sección de este manual) son bastante claras de comprender tanto para la escritura de código como para la lectura con el objetivo de evitar ambigüedades. Esto hace que no necesites ser un experto para entender y crear un nuevo código.

## **Uniformidad.**

*WareLang* mantiene consistencia en sus estructuras de control, dejando a las palabras clave como principal diferenciador entre las estructuras y sin verse en la necesidad de alterar su estructura para las palabras clave especificadas, ya sean las que definen las estructuras de control, las palabras clave para definir un tipo de dato, operadores y demás palabras propias del lenguaje. Pese a que la estructura se compara con la programación orientada a objetos no está diseñada para este paradigma de programación.

## **Ortogonalidad.**

En *WareLang* se contempla que las características se combinan de manera coherente. Las palabras reservadas se pueden combinar entre sí sin causar errores, pero de una manera coherente.

Se aplica la ortogonalidad en los operadores y tipos de datos, entre los operadores se pueden combinar de manera flexible y con coherencia. Un ejemplo se combinan operadores aritméticos con operador de asignación.

Ocurre en *WareLang* la combinación de controles de flujo (bucles y condiciones) y estructura de datos (arreglos y listas). Ejemplo de una combinación es utilizar un bucle para recorrer un arreglo.

## Diseño Detallado.

### Alfabeto

*WareLang* cuenta con el siguiente Alfabeto:

Caracteres	A B C D E F G H I J K L M N Ñ O P Q R S T U V X Y Z a b c d e f g h i j k l m n ñ o p q r s t u v x y z space tab salto de línea ¡ ? :
Símbolos	& ! = < > ! () [] {} ; ' ' = < > ! + * - % , . / ^ "
Operadores relacionales	= < > !
Operadores aritméticos y asignación	+
Dígitos	0 1 2 3 4 5 6 7 8 9
Operadores de agrupación y delimitación	() [] {} ; ' '
Símbolo de puntuación	, .
Operadores lógicos	& !

### Tipos de datos.

*WareLang* es un lenguaje tipado, cada variable debe tener un tipo de dato específico y no puede cambiar de tipo durante la ejecución del programa. Esto significa que debemos declarar el tipo de cada variable antes de usarla y asegurarnos de que los valores asignados sean compatibles con ese tipo.

### PUERTO

Tipo de dato para definir puertos. Los puertos son todos los dispositivos externos al robot que incluyen sensores, pantallas, y otros componentes a los que se necesite mandarles una señal.

### VOL

Tipo de dato para definir el volumen de la alarma.

## **ENT**

Es un tipo de dato de 32 bits con signo para almacenar valores numéricos. Cuyo valor mínimo es -2147483648 y el valor máximo 2147483647.

- Valor por defecto: 0

## **COLOR**

Es un tipo de dato que guardará un color para el sensor de color, sera un dato que guardará el valor RGB, del color.

## **DEC**

Es un tipo de dato de coma flotante de precisión simple de 64 bits.

- Valor por defecto: 0.0

## **BOOL**

Sirve para definir tipos de datos booleanos. Es decir, aquellos que tienen un valor de VERDADERO o FALSO (1 o 0). Ocupa 1 bit de información.

- VERDADERO = 1
- FALSO = 0
- Valor por defecto: FALSO

## **CAD**

Permite manejar cadenas de caracteres delimitadas por comillas simples, es un tipo de dato que soporta la concatenación.

- Valor por defecto: NULL.
- NULL también se puede identificar como ''.

## **Grupos de Cadenas.**

### **Operadores relacionales.**

Los operadores relacionales son símbolos utilizados en programación para comparar dos valores y determinar la relación entre ellos. Estos operadores devuelven un valor booleano (verdadero o falso) que indica si la relación especificada es verdadera o no.

Igualdad	==
----------	----

Desigualdad	$!=$
Mayor que	$>$
Menor que	$<$
Mayor o igual que	$\geq$
Menor o igual que	$\leq$

## Operadores aritméticos y asignación.

Los operadores aritméticos son operadores que sirven para realizar operaciones matemáticas con valores numéricos, tanto enteros como reales, los operadores aritméticos en este lenguaje son los siguientes:

Suma	$+$
Resta	$-$
División	$/$
Multiplicación	$*$
Módulo	$\%$
Potencia	$^$

Por otro lado, los operadores de asignación, como su nombre sugiere, sirven para darle cierto valor a las variables, algunos de ellos pueden realizar una operación previo a asignar el valor. Los operadores de asignación presentes en *WareLang* son los siguientes:

Asigna un valor directo a una variable	$=$
Le suma al valor de la variable actual el valor de la derecha	$+=$
Le resta al valor de la variable actual el valor de la derecha	$-=$
Divide el valor de la variable actual por el valor de la derecha	$/=$
Almacena el residuo de la división de la variable actual entre el operador a la derecha.	$\% =$

## Números Enteros.

Es una combinación de 1 o más dígitos que pueden ser positivos o negativos representan un valor numérico.

Positivos	Negativos
+223232	-232
2323442	-1
+948782	-9384
23	-4
9	-6635

## Números Reales.

Es una combinación de 1 o más dígitos, seguidos de un punto, seguido de por lo menos 1 o más dígitos que pueden ser positivos o negativos representan un valor numérico.

Positivos	Negativos
22.3232	-2.32
+23234.42	-1.0
94.8782	-9.384
+2.3	-4.0
9.0	-66.35

## Símbolos de agrupación y delimitación.

Se utilizan para agrupar operaciones y establecer un orden específico en su ejecución.

Paréntesis	( )
Corchetes	[ ]
Llaves	{ }

## Operadores lógicos.

Con estos se puede realizar operaciones lógicas con tipos de datos booleanos.

Y	&&
O	
No	!

## Símbolos de puntuación.

Dentro de *WareLang* tenemos los siguientes símbolos de puntuación, los cuales tienen distintos significados sintácticos y semánticos para el compilador:

Punto y coma	;
Punto	.
Coma	,
Guion bajo	_
Dos puntos	:

## Comentarios

Los comentarios sirven como documentación del programa. Hay dos formas:

- Los comentarios de línea comienzan con la secuencia de caracteres // y terminan al detectar un salto de línea (Enter).
- Los comentarios generales comienzan con la secuencia de caracteres /\* y terminan con la primera secuencia de caracteres \*/ subsiguiente.

## Identificadores.

Los identificadores *WareLang* son un espacio de memoria en el que guardamos un determinado valor (o dato). Tiene un límite de 30 caracteres. Los identificadores deben iniciar con una letra minúscula obligatoriamente seguido de 1 o más letras o dígitos, como se muestra a continuación:

Identificador válido	Identificador no válido
<code>ident122;</code> <code>suma;</code> <code>a;</code>	<code>1;</code> <code>\$_var;</code> <code>Suma;</code>

*WareLang* es un lenguaje de tipado, por lo cual a todas las variables se le deberá especificar su tipo de dato para ser utilizadas. El tipo de dato se asignará a la hora de definir la variable, de igual forma se puede agregar un valor por default. Como se muestra a continuación:

```

ENT distancia;
CAD cadena1 = 'Vacio';

```

## Palabras Reservadas

Las siguientes palabras clave están reservadas y no pueden ser utilizadas como identificadores, estas deben escribirse con letras mayúsculas, siguiendo su patrón de caracteres.

### Palabras reservadas para la estructura y flujo del programa.

SI	CAD
SINO	CLASE
FALSO	DEF
REPETIR	IMPR
RETORNA	MIENTRAS
DEC	VERDADERO
BOOL	CONSOL

### Palabras reservadas para el funcionamiento del robot.

LEER	PARAR
ESCRIB	TOMAR
ADELANTE	SOLTAR
ATRAS	CAJA
IZQUIERDA	REVISAR
DERECHA	LCD
ALARMA	PRENDER
APAGAR	LIMPIAR
SUBIR	BAJAR

## **Microestructura.**

### **Estructura de las expresiones.**

Dentro de WareLang para el manejo o definición de las expresiones, debemos tener en cuenta los siguientes puntos al momento de realizarlas. Una expresión representa la operación entre uno o más operadores, siendo estas funciones, métodos, variables o valores numéricos, o cadenas de caracteres.

#### **Expresiones de concatenación de cadenas:**

'Hola' + nombre

En esta expresión, se concatena la cadena "Hola" con el contenido de la variable nombre.

#### **Delimitacion del programa y métodos.**

Los corchetes son utilizados para delimitar el inicio y final de un método, programa o bloque de código. Su sintaxis es la siguiente:

```
CLASE Programa 1{ // Inicio del programa
REPETIR(2) { // Inicio de la función REPETIR
    IZQUIERDA(1, 3);
    DERECHA(4, 3);
} // Fin de la función REPETIR
}// Fin del programa
```

#### **Delimitación de sentencias y líneas de código.**

El punto y coma (;) delimita las expresiones y líneas de código. Representa el final de una sentencia o conjunto de sentencias. Su sintaxis es la siguiente:

```
DEC ident1 = 134.02; // Finalización de sentencia
DERECHA(4, 3); // Finalización de sentencia
```

## Definición de un método con DEF.

La definición de un método en *WareLang* se realiza dentro de la definición de una clase. Un método es una función que el usuario puede definir, el cual sigue la siguiente estructura:

```
DEF nombre_del_método (parámetros) {  
    //En esta parte se escribe el código  
}
```

Entre paréntesis se escriben los parámetros necesarios para realizar operaciones específicas, un método puede tener o no parámetros. Para llamar un método se escribe el nombre del método, paréntesis y si los hay, parámetros, como a continuación:

```
nombre_del_método(); //De esta forma llamamos al método  
                      //en otra línea, después de declararlo.
```

## Métodos del lenguaje.

WareLang cuenta con ciertos métodos propios para realizar acciones o determinar movimientos o comportamientos del robot. En las siguientes secciones se muestran la definición de estos métodos, su sintaxis y el valor que deberían de tener sus parámetros.

### Método ADELANTE()

Esta palabra clave, reservada en el contexto del control de un robot, indica que el robot debe avanzar hacia adelante. No retorna valores y admite los siguientes parámetros para controlar el movimiento y la velocidad del robot.

```
ADELANTE ( espacios que avanza , velocidad del robot );
```

- *Espacios que avanza*, es la cantidad de cajas que avanzará el robot, y debe ser tipo ENT, el valor predeterminado es 0.
- *La velocidad del robot*, es de tipo ENT, admite los siguientes valores:
  - 1 para velocidad baja
  - 2 para velocidad media
  - 3 para velocidad alta
  - El valor predeterminado es 2.

### **Método ATRAS().**

Esta palabra reservada representa la instrucción que le indica al robot que avance hacia atrás en relación a su posición actual. No retorna valores y admite los siguientes parámetros para controlar el movimiento y la velocidad del robot.

**ATRAS** (espacios que avanza, velocidad del robot);

- *Espacios que avanza*, es la cantidad de cajas que avanzará el robot, y debe ser tipo **ENT**, el valor predeterminado es 0.
- *La velocidad del robot*, es de tipo **ENT**, admite los siguiente valores:
  - 1 para velocidad baja
  - 2 para velocidad media
  - 3 para velocidad alta
  - El valor predeterminado es 2.

### **Método IZQUIERDA().**

Esta palabra reservada representa la instrucción que le indica al robot que avance hacia la izquierda en relación a su posición actual. No retorna valores y admite los siguientes parámetros para controlar el movimiento y la velocidad del robot.

**IZQUIERDA** (espacios que avanza, velocidad del robot);

- *Espacios que avanza*, es la cantidad de cajas que avanzará el robot, y debe ser tipo **ENT**, el valor predeterminado es 0.
- *La velocidad del robot*, es de tipo **ENT**, admite los siguiente valores:
  - 1 para velocidad baja
  - 2 para velocidad media
  - 3 para velocidad alta
  - El valor predeterminado es 2.

### **Método DERECHA().**

Esta palabra reservada representa la instrucción que le indica al robot que avance hacia la izquierda en relación a su posición actual. No retorna valores y admite los siguientes parámetros para controlar el movimiento y la velocidad del robot.

**DERECHA** (espacios que avanza, velocidad del robot);

- *Espacios que avanza*, es la cantidad de cajas que avanzará el robot, y debe ser tipo **ENT**, el valor predeterminado es 0.
- *La velocidad del robot*, es de tipo **ENT**, admite los siguiente valores:
  - 1 para velocidad baja
  - 2 para velocidad media
  - 3 para velocidad alta
  - El valor predeterminado es 2.

### **Método PARAR().**

Esta palabra reservada representa la instrucción que indica al robot que se detenga en su posición actual. No lleva parámetros y tampoco retorna valores.

**PARAR();**

### **Método TOMAR().**

Esta palabra reservada representa la instrucción que indica al robot tomar el contenido de la caja sobre la que se encuentra en ese momento. No lleva parámetros y no retorna valores.

**TOMAR();**

### **Método SOLTAR().**

Esta palabra reservada representa la instrucción que indica al robot que deje o suelte el contenido que lleva consigo en ese momento. No lleva parámetros y no retorna valores.

**SOLTAR();**

### **Método REVISAR().**

Esta palabra reservada representa la instrucción con la cual el robot revisará si existen o no artículos dentro de la caja sobre la que se encuentra el robot en ese momento, retornando un valor booleano, siendo *verdadero* si encuentra contenido o *falso* si no encuentra contenido. Esto por medio de un sensor de color, el cual determina el color de fondo y si el color que detecta es ese, entonces determina que esta vacío.

**REVISAR(color, puerto);**

- COLOR. Valor hexadecimal del color que debe buscar para que coincida.

### Método CAJA().

Esta palabra sirve para definir el tamaño de las cajas sobre las que se moverá el robot. Al definir las medidas de las cajas se define la distancia que se deberá desplazar el robot para pasar de una a otra. No retorna valores y los parámetros que lleva y su uso se ven a continuación:

```
CAJA(medida de la caja);
```

- *Medida de la caja.* La medida de la caja en centímetros, es un número decimal.

### Método IMPR().

Esta palabra reservada permite hacer una impresión en consola. Se pueden imprimir tanto mensajes con el uso de comillas simples como imprimir variables.

```
//Este programa imprimirá la palabra "Texto" y "El valor es: 5"
CLASE impresion{
    // Impresión con uso de comillas simples.
    IMPR('Texto', CONSOLE);

    //Impresión con uso de variables.
    ENT variable = 5;
    IMPR('El valor es: ' + variable , CONSOLE);
}
```

## Estructuras de datos.

El lenguaje *WareLang* cuenta con ciertas estructuras de datos con las cuales maneja el espacio y la lista de objetos que debe buscar el robot, para estos propósitos se usarán tanto las matrices como las listas, cuyo uso y funcionalidad se explica a continuación.

### Matrices.

Las matrices son importantes en *WareLang* para el guardado de posiciones de los diferentes objetos o espacios que podría tener el almacén, con el fin de automatizar traslados, utilizando únicamente el nombre del producto. Las matrices en *WareLang* se declaran con la siguiente estructura:

```
TIPO_DE_DATOS NombreMatriz [Núm. de Renglones] [Núm. de Columna] ;
```

1. Para ingresar los datos lo podemos realizar de dos formas . Inicialización explícita.

```
ENT NombreMatriz = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

2. Asignación de valores después de la inicialización:

```
ENT matriz1 [5] [5] ;  
NombreMatriz [0] [1] = 64;  
NombreMatriz [0] [2] = 99;
```

### Listas enlazadas.

Dentro de *WareLang* las listas existen como una clase propia del lenguaje, las cuales se declaran y sirven para añadir objetos los cuales el robot debe buscar dentro del área de búsqueda. La lista también cuenta con ciertos métodos para agregar o eliminar elementos de la lista.

```
TIPO_DE_DATO Nombre_Lista = [] ;
```

### Por ejemplo:

```
DEC lista1 = [] ;
```

También se puede inicializar con ciertos valores al momento de la declaración:

```
CAD frutas = ['mango', 'fresa', 'ciruela', 'chabacano'] ;
```

De esta manera se crea la lista vacía, pero se pueden agregar los campos u objetos que serán parte de la lista desde el momento de la declaración. Principalmente se usarán 2 métodos en conjunto con la lista:

- Lista.agregar(objeto): añade el objeto colocado entre paréntesis al contenido de la lista, no retorna nada
- Lista.sacar([x]): el parámetro x es opcional, si no se especifica, saca de la lista el último elemento y retorna, si se especifica, buscará en el contenido de la lista y si no lo encuentra o si está vacía, retorna un valor nulo.

## Estructuras de control.

### Estructuras Secuenciales.

*WareLang* ejecuta sus instrucciones de manera secuencial, siguiendo el orden en que están escritas, siempre y cuando no exista ningún otro tipo de estructura de control dentro del programa.

### Estructuras Selectivas.

#### Sentencia SI

La sentencia SI en *WareLang* nos permite ejecutar un bloque de código si una condición dada es verdadera o falsa.

```
SI (condición) {  
    // Aquí se escribe el código si cumple la condición  
}  
  
SINO {  
    // Aquí se escribe el código si NO cumple la condición  
}
```

Estructura selectiva doble tiene la palabra clave ‘SINO’. Cuando la condición no es verdadera entra el ‘SINO’, indicando a ejecutarse.

#### Ejemplo:

```
CLASE usoSI{  
    SI (REVISAR () == TRUE) {  
        TOMAR ();  
        IZQUIERDA (2,1);  
    }  
    SINO {  
        IMPR ('CAJA VACIA', CONSOLE);  
    }  
}
```

## Estructuras Iterativas o Repetitivas.

### Sentencia MIENTRAS

Este bucle ejecuta un bloque de código mientras una condición específica sea verdadera. La condición se evalúa antes de cada iteración. Esta sentencia sigue la siguiente estructura:

```
MIENTRAS (condición) {  
    //Bloque de código  
}
```

Ejemplo:

```
MIENTRAS (ALARMA () == VERDADERO) {  
    IMPR ('Encendido', CONSOL);  
}
```

### Sentencia REPETIR

Este bucle ejecuta un bloque de código un número específico de veces. Puede iterar sobre una secuencia, como una lista o un rango de números. Esta sentencia sigue la siguiente estructura:

```
REPETIR(número de veces que se repite) {  
    //Bloque de código  
}
```

Ejemplo:

```
CLASE ejemploRep{  
    ENT cVacias = 0;  
    ENT cLlenas = 0;  
    REPETIR(4) {  
        ADELANTE(1);  
        SI (REVISAR () == VERDADERO) {  
            cLlenas++;  
        } SINO{  
            cVacias++;  
        }  
    }  
    IMPR ('Cajas llenas: '+cLlenas,LCD);  
    IMPR ('Cajas vacias: '+cVacias,LCD);  
}
```

## **Estructuras de compilación.**

Para cualquier lenguaje de alto nivel, se necesita un proceso de traducción hacia un lenguaje de máquina comprensible. Esto se logra mediante compiladores. Un compilador es un programa que convierte código escrito en un lenguaje de alto nivel a lenguaje de máquina. Es esencial que el compilador mantenga la precisión y velocidad de compilación, conserve el significado del código original, genere código objeto eficiente, reconozca construcciones válidas e inválidas, maneje errores adecuadamente y permita la depuración del código.

### **Análisis léxico.**

La fase de análisis léxico es la primera etapa del proceso de compilación. Durante esta fase, el compilador escanea el código *WareLang* y lo divide en tokens, que son unidades léxicas significativas como palabras clave, identificadores y otros componentes léxicos. Estos tokens se clasifican en diferentes clases y se almacenan en tablas para su uso posterior. Además, el análisis léxico ignora los comentarios en el código y detecta cualquier token que no pertenezca al lenguaje.

### **Análisis Sintáctico.**

El analizador sintáctico identifica la estructura del código creado en lenguaje *WareLang* y verifica si sigue las reglas del lenguaje. Utiliza tokens generados por el analizador léxico para construir un árbol de análisis. Sus funciones principales incluyen obtener tokens, verificar la sintaxis, reportar errores y generar la estructura jerárquica del árbol de análisis.

### **Análisis Semántico.**

El análisis semántico verifica que el código tenga sentido semánticamente, utilizando el árbol de sintaxis y la tabla de símbolos. Se asegura de que los tipos y operadores se utilicen correctamente, almacenando información de tipo, verificando tipos, y recopilando datos sobre compatibilidad.

### **Generación de código intermedio.**

La fase de generación de código intermedio se encarga de crear una representación intermedia del código en *WareLang*, que está entre el lenguaje de alto nivel y el código de máquina. Su función principal es facilitar la traducción al lenguaje de destino, conservando su significado y estructura original.

### **Optimización del código.**

En esta etapa, se optimiza el código para mejorar su velocidad y eficiencia. Se eliminan líneas innecesarias y se reorganizan las declaraciones para mejorar la ejecución del programa, reducir el uso de recursos y generar un código más rápido y compacto.

### **Generación de código.**

La última fase del compilador se encarga de tomar la entrada del optimizador de código y producir código objeto o código máquina. Aquí, las instrucciones en código intermedio se traducen al lenguaje máquina, y se asignan ubicaciones de memoria para las variables y otros datos del programa, así como la gestión de la asignación de memoria y registros.

### **Tablas de símbolos.**

La tabla de símbolos es un registro de identificadores con información como los tipos de datos existentes en *WareLang* (ENT, CAD, DEC, BOOL, VOL) y otros atributos, como palabras reservadas (ADELANTE, ATRÁS, SI, MIENTRAS). Esta tabla facilita la recuperación rápida de información durante el proceso de compilación y se actualiza continuamente con la interacción del manejador de errores en todas las fases del proceso.

### **Manejador de errores.**

El compilador de *WareLang* cuenta con una tabla de errores previamente declarados para poder identificarlos dentro del código. El manejo de errores debe proporcionar información clara al programador sobre el tipo de error y su ubicación en el código.

## Estructura del programa.

```
//Inicio del programa
CLASE programaEjemplo{

//Declaración de variables y asignaciones
    VOL num1 = 200;
    DEC num2 = 0.2;
    CAD palabra;
    PESO pesoJitomate;

//Declaración de métodos
DEF pesarJitomate () {
    pesoJitomate = PESAR(KILOS);
    RETORNA pesoJitomate;
}

//Estructuras de control

SI (REVISAR())== TRUE) {
    pesarJitomate(); //Llamar al método
}
SINO {
    IMPR ('CAJA VACIA', CONSOLE);
}

}//Fin del programa
```

## Estructuras de la entrada/salida.

### Sensores y Dispositivos.

- **Sensor de color:** Un robot equipado con un sensor de color puede realizar la tarea de detectar la presencia de artículos en cajas de manera eficiente. A medida que el robot se aproxima a una caja, el sensor de color escanea el área y registra los colores presentes. Los valores de color detectados se comparan con los criterios predefinidos para determinar si hay artículos dentro de la caja. Dependiendo del resultado de la detección, el robot toma diferentes acciones.

- **Sensor de Proximidad:** Un robot equipado con un sensor de proximidad es capaz de detectar qué tan cerca de algún objeto o pared. Es muy útil en caso que no se haya definido de manera correcta las dimensiones donde el robot se va a movilizar.
- **Motores de corriente continua (DC):** Estos son los motores más comunes para robots con ruedas. Son fáciles de controlar y están disponibles en una variedad de tamaños y velocidades. Se pueden controlar fácilmente la velocidad y dirección de giro cambiando la polaridad de la alimentación eléctrica.
- **Bocina:** El robot está equipado con una bocina que emite un sonido al detectar una señal o al ser activada por cierto evento. El sonido de la bocina está pensado para alertar de algún fallo o mal funcionamiento al momento de operación. La desactivación de la misma será al momento de que se detecte que el evento que la activó primero ha sido atendida o manualmente por el supervisor del sistema
- **Pinzas mecánicas:** Funcionan mediante un mecanismo de apertura y cierre controlado, que permite al usuario abrir y cerrar las mandíbulas de la pinza para agarrar y soltar objetos. El movimiento de apertura y cierre de las mandíbulas de la pinza puede ser controlado manualmente por el usuario o automáticamente mediante un sistema de actuadores, como un servomotor o un motor paso a paso.
- **Computadora:** Por medio de una computadora, tenemos la capacidad de ingresar datos y recibir información. Este proceso es esencial para el desarrollo de código, ya que permite tanto la entrada de parámetros y variables como la salida de resultados y mensajes. La computadora actúa como una interfaz entre el programador y el sistema en el que se está trabajando.