NOMBRE DEL COMPILADOR

SPINDRO

ESTE COMPILADOR SERÁ SOBRE UN LENGUAJE PARA MANIPULAR UN PEQUEÑO TALADRO POR MEDIO DE ARDUINO.

OBJETIVO:

PODER MANIPULAR UN TALADRO POR MEDIO DEL USO DE PROGRAMACIÓN, MANDANDO INSTRUCCIONES DESDE UN LENGUAJE PROPIO AL MINI TALADRO.

FUNCIONES BÁSICAS

ATORNILLAR Y DESATORNILLAR CON LA VELOCIDAD QUE EL USUARIO ESCOJA, APAGARSE.

LOGOTIPO

Tabla de contenido

Gramática del compilador

No.	Tipo	Tokens
1	Palabras reservadas	clase
2	Identificadores	(Cualquier palabra que comience con una letra ya sea minúscula o mayúscula) Ejemplos: Palabra, palabra, palabra3, …
3	Símbolos	=, +, -, *, /, (,)
4	Apertura y cierre de bloque	<, >
5	Tipo de dato	numero, verificar
6	Comentarios	//
7	Separador de tokens	;
8	Valor	(números del 0 al 9 en cualquier cantidad), verdadero, falso
9	Expresión de instrucciones	atornillar(número), desatornillar(número), apagar(), tiempo(número)
10	Expresión de asignación	Identificador=valor
11	Expresión	(Identificadores, valores, tipo de dato e identificador)

Instrucciones

1. atornillar: Hace girar el motor hacia la izquierda con la velocidad que se quiera realizar.
2. desatornillar: Hace girar el motor hacia la derecha con la velocidad que se quiera.
3. apagar: Detiene el motor.
4. tiempo: Continúa la última instrucción durante los segundos indicados.

Ejemplo de código

clase owo

<

numero uwu = 2; //Epico 7w7

atornillar(uwu);

desatornillar(5);

tiempo(10);

apagar();

>

Análisis léxico

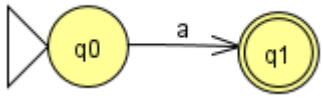
Mi lenguaje para mi compilador tendrá una estructura similar a c# y otros lenguajes de programación ya existentes, tendrá varios elementos y gramática como: Identificadores, apertura y cierre de bloques, expresiones de asignación, valores, parámetros, tipos de datos, palabras claves, separadores de tokens, instrucciones y condiciones.

Expresiones regulares

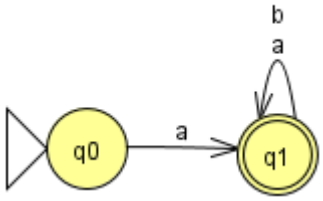
1. Palabras reservadas: **a** //*donde a=clase* | *funcion* | *si* | *sino*
2. Identificadores: **aa*b*** //*donde a=cualquier letra y b=cualquier número*
3. Símbolos: **a** //*donde a= (= | + | - | * | / | (|))*
4. Apertura y cierre de bloques: **a** //*donde a*= < | >
5. Tipo de dato: **a** //*donde a= numero* | *decimal* | *letra* | *verificar*
6. Comentarios: ***//aa**** //*donde a= (cualquier letra, número o símbolo)*
7. Separador de tokens: **;**
8. Valor: **([0-9][0-9]*), (Identificador), (verdadero), (falso)**
9. Expresión de instrucción: **atornillarExpresión;, desatornillarExpresión; , tiempoExpresión; , apagar();**
10. Expresión de asignación: **Identificador=valor, Identificador=Identificador**
11. Expresión: **(Identificador), (valor), (TipoDato identificador)**

Autómatas

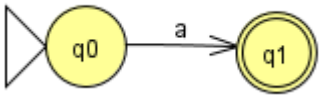
Palabras reservadas:



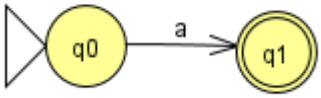
Identificadores:



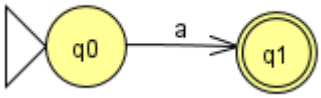
Símbolos:



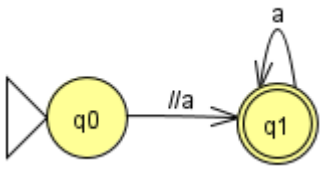
Apertura y cierre de bloques:



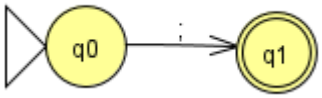
Tipo de dato:



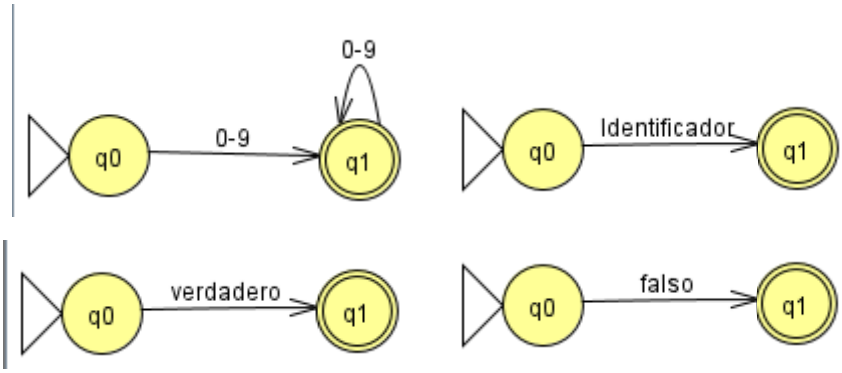
Comentarios:



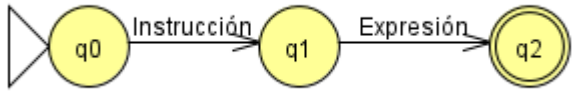
Separador de tokens



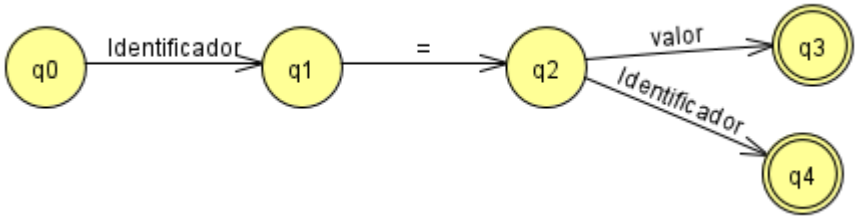
Valor:



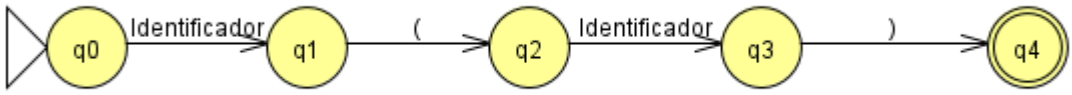
Expresión de instrucción:

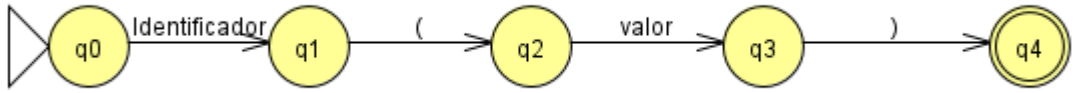


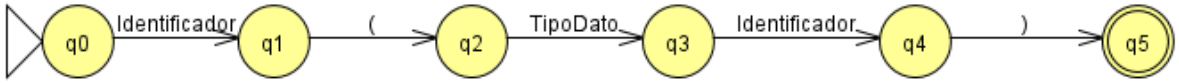
Expresión de asignación:



Expresión:







Matrices de estados

Palabras reservadas	a
Q0	Q1
Q1	

Identificador	a	b
Q0	Q1	
Q1	Q1	Q1

Símbolos	a
Q0	Q1
Q1	

Apertura y cierre de bloques	a
Q0	Q1

Q1	

Tipo de dato	a
Q0	Q1
Q1	

Comentarios	//a	a
Q0	Q1	
Q1		Q1

Separador de tokens	;
Q0	Q1
Q1	

Valor 1	0-9
Q0	Q1
Q1	Q1

Valor 3	Identificador
Q0	Q1
Q1	

Valor 4	verdadero
Q0	Q1
Q1	

Valor 5	falso
Q0	Q1
Q1	

Expresión de instrucción	Instrucción	Expresión
Q0	Q1	
Q1		Q2
Q2		

Expresión de asignación	Identificador	=	valor	Identificador
Q0	Q1			
Q1		Q2		
Q2			Q3	

Q3				Q4
Q4				

Expresión 1	Identificador	(	)
Q0	Q1		
Q1		Q2	
Q2	Q3		
Q3			Q4
Q4			

Expresión 2	Identificador	(	valor	)
Q0	Q1			
Q1		Q2		
Q2			Q3	
Q3				Q4
Q4				

Expresión 3	Identificador	(	TipoDato	)
Q0	Q1			
Q1		Q2		
Q2			Q3	
Q3	Q4			
Q4				Q5
Q5				

Análisis sintáctico

Para mi lenguaje se utilizarán 5 expresiones, las cuales son las siguientes: Instrucciones, clases, métodos, asignaciones y condiciones. Las cuales tendrán su propia sintaxis, y errores que se encontrarán al no cumplirse el orden definido.

Sintaxis de instrucciones

1. **Expresión de instrucción:** Instrucción-Expresión-SeparadorDeTokens

   //Ejemplo: subir(5);

   //Para apagar

Instrucción-()-SeparadorDeTokens

   //Ejemplo: apagar();

2. **Declaración de clases:** clase-Identificador

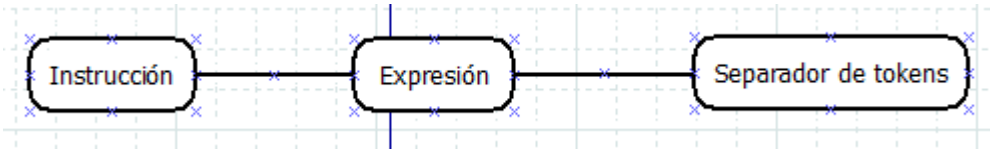
   //Ejemplo: clase ola

3. **Expresión de asignación:** TipoDeDato – Identificador = Valor-SeparadorDeTokens

   //Ejemplo: numero numero = 10;

Árboles sintácticos

Expresión de instrucción:



Declaración de clase:

Expresión de asignación:

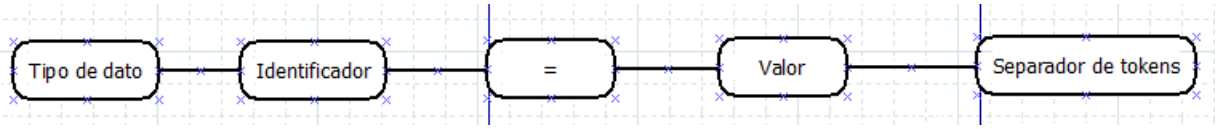


Tabla de errores sintácticos

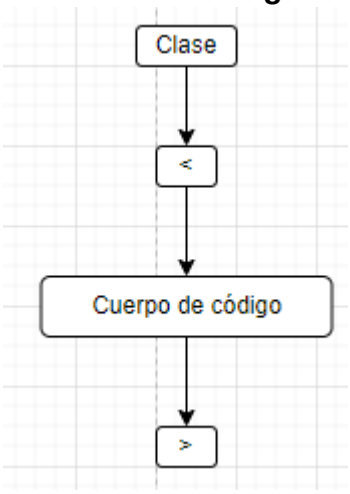
No.	Expresión	Errores
1	Expresión de instrucción	• Falta expresión para la instrucción. • Falta sobrecarga para la expresión de la instrucción. • Falta instrucción para la expresión o sobrecarga. • Falta separador de tokens “;” para la expresión de instrucción.
2	Declaración de clase	• Falta tipo de dato o definición para el identificador. • Falta identificador para la clase. • La declaración de una clase no debe llevar separador de tokens “;”.
3	Expresión de asignación	• Falta tipo de dato o definición para el identificador. • Falta identificador para el tipo de dato. • Falta símbolo de “=” en la expresión de asignación. • Falta asignar un valor a la variable. • Falta separador de tokens en la expresión de asignación “;”. • Falta definir la variable en el símbolo “=”.

Análisis Semántico

El análisis semántico se encargará de darle sentido al código que se está escribiendo, en este caso dando el formato en el que redactar un código funcional y con sentido.

Más específicamente, dictando que va antes o después de cierto código escrito, y lo que se necesita para que este sea funcional, además de darle orden a las instrucciones y darles una cierta estructura para que funcionen correctamente.

Árbol semántico general del compilador

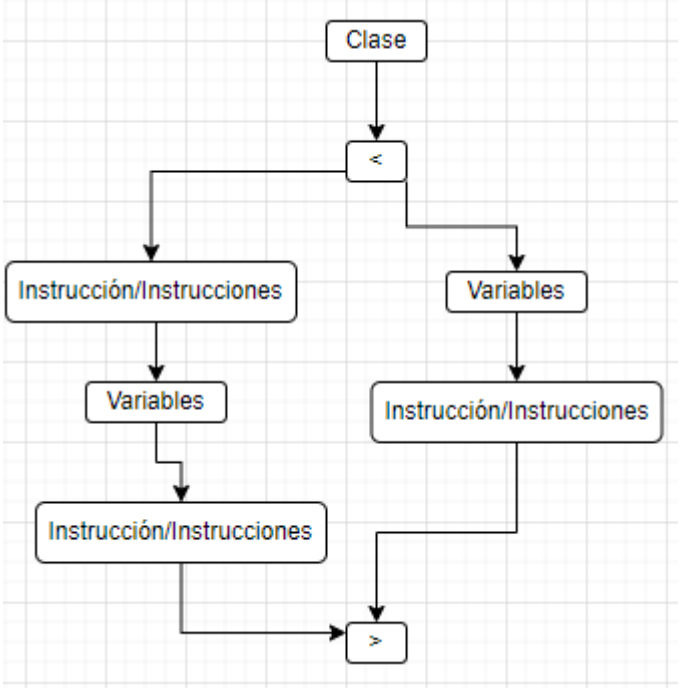


//Donde Cuerpo de código pueden ser instrucciones o //declaraciones de variables, o comentarios

Reglas del programa (R(p))

No	Regla	Error	Mensaje
1	Se debe iniciar con “clase”	No se ha declarado una clase	Declarar la clase
2	Sólo se puede declarar una clase	Sólo se puede declarar una clase	Eliminar clase de más
3	Después de declarar una clase se debe colocar la apertura de bloque “<”	Se esperaba apertura de bloque “<”	Colocar <
4	La última línea debe ser “>”	Se esperaba cierre de bloque “>”	Colocar >
5	Sólo existe un “<”	Sólo debe existir una apertura de bloque “<”	Mantener un solo “<”
6	Sólo existe un “>”	Sólo debe existir un cierre de bloque “>”	Mantener un solo “>”
7	La última sentencia antes del cierre de bloque debe ser una instrucción	Falta instrucción de apagado al final del código	Colocar la instrucción de apagar
8	Los valores de una variable tienen que ser distintos a la propia variable	No se puede asignar de valor a la variable la propia variable	Colocar un valor válido
9	Cada variable usaba debe haber sido declarado anteriormente	No se puede usar una variable que no se ha declarado	Colocar una variable o calor válido

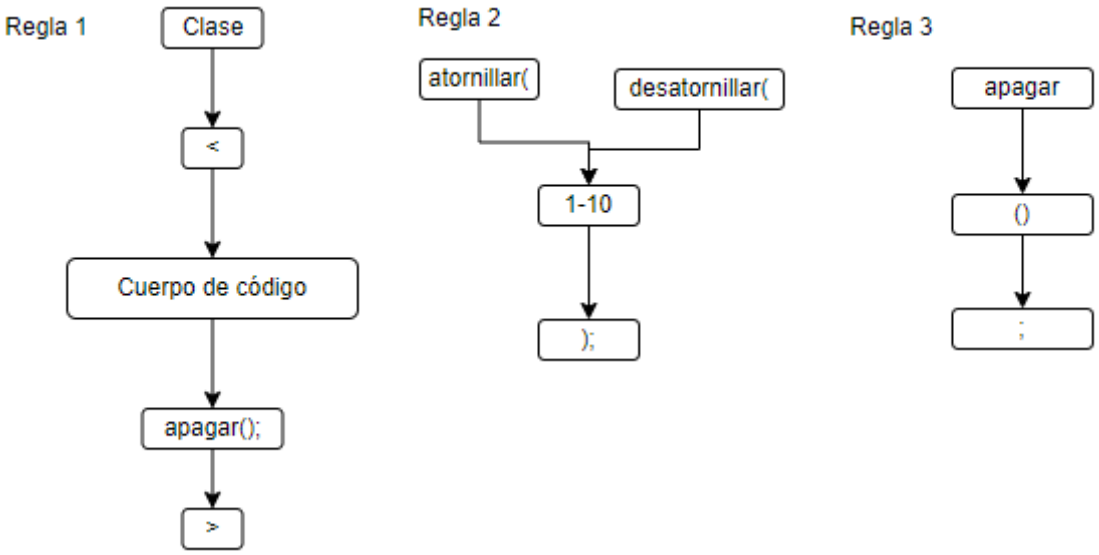
Árbol semántico de reglas del programa



Reglas de Instrucción (r(a))

No	Regla	Error	Mensaje
1	La instrucción apagar debe ser la última instrucción	Debe haber una instrucción de apagar al final de código	Colocar la instrucción de apagar al final del código.
2	Las instrucciones de atornillar y desatornillar deben llevar de parámetro un valor mínimo de uno, hasta un valor máximo de diez	La instrucción debe llevar un valor entre 1 y 10	Colocar un valor del 1 al 10
3	La instrucción de apagar se redacta sin parámetros	La instrucción de apagar no debe llevar parámetros	Eliminar los parámetros de la instrucción

Árboles semánticos de las reglas de instrucción



/*Tabla de traducción

Código	Significado	Función

clase	void setup()
	{
	pinMode(pin1,OUTPUT);
	pinMode(pin2,OUTPUT);
	}
		Declarar los pines y el comienzo del código.

<	void loop() {	Apertura de código.
>	}	Cierre de código.
apagar();	digitalWrite(pin1,0); digitalWrite(pin2,0);	Apagar el taladro.
atornillar(num);	analogWrite(pin1,(num*12)+100); digitalWrite(pin2,0);	Gira el taladro hacia la izquierda, con la velocidad seleccionada.
desatornillar(num);	analogWrite(pin2(num*12)+100); digitalWrite(pin1,0);	Gira el taladro hacia la derecha con la velocidad seleccionada.
tiempo(num);	delay(num*1000);	Continúa la última instrucción durante los segundos indicados.
numero num = valor;	int num = valor;	Declaración de una variable tipo int.
+, -, /, *	+, -, /, *	Signo de suma, resta, división y multiplicación.
//comentario	//comentario	Comentarios

***/**