



COMPILADORES 1

HASHTAG

MANUAL DE USUARIO

ING. CARLOS VALLEJO

INTEGRANTES:

STEPHANIE SCHOENHERR	1094-1150
WILMER CARRANZA	1101-1227

24 DE MARZO DE 2015

Índice

INTRODUCCIÓN HASHTAG	3
HASHTAG - COMPILER	4
PROGRAMANDO EN HASHTAG	6
IDENTIFICADORES	7
TIPOS DE DATOS.....	7
DEFINICIÓN E INICIALIZACIÓN DE IDENTIFICADORES	8
PALABRAS RESERVADAS	9
OPERADORES	10
COMENTARIOS.....	11
FUNCIONES HASHTAG.....	11
LLAMADOS	12
PRINTS.....	12
ESTRUCTURAS DE CONTROL DEL FLUJO	13
IF	13
SWITCH.....	14
WHILE	15
FOR	15
SENTENCIA RETURN	15
DESARROLLO HASHTAG - COMPILER MEDIANTE JFLEX Y CUP.....	16
JFlex.....	16
Analizador Léxico.....	16
CUP	20
Analizador Sintáctico	20
EJEMPLO PROGRAMA COMPLETO HASHTAG.....	26

INTRODUCCIÓN HASHTAG

Hashtag es un lenguaje de programación fácil de aprender. Cuenta con su propia estructura y reglas de sintaxis. La elegante sintaxis permite al desarrollador crear libremente su código para formar un programa.

Los programas en Hashtag son ordenados y legibles, donde se puede visualizar con claridad la creación de funciones, instrucciones, bloques de código etc.

Este manual introduce de manera general al lector los conceptos y características básicas del lenguaje Hashtag.

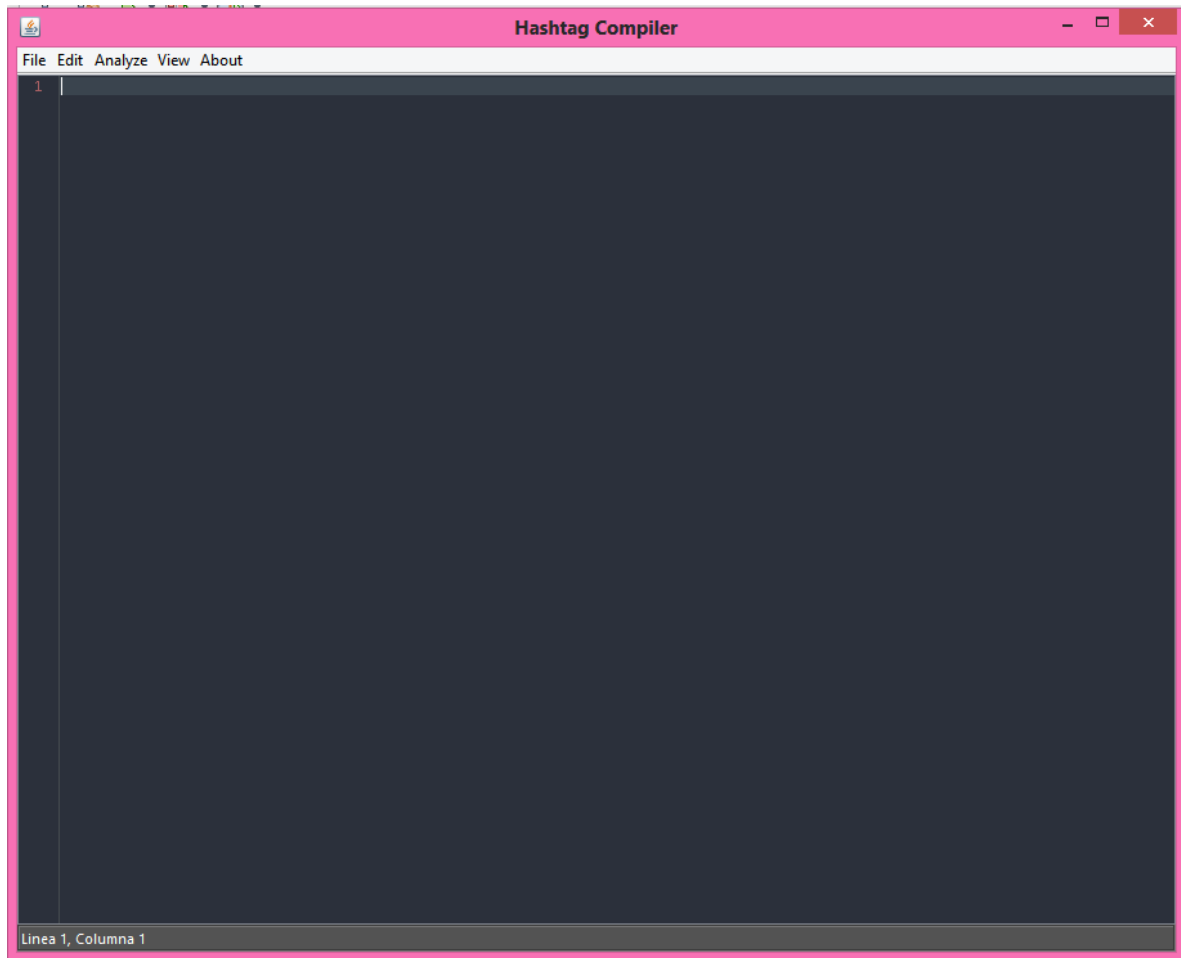
Los elementos que contiene son los siguientes:

- Identificadores
- Tipos de Datos
- Palabras Reservadas
- Comentarios
- Operadores
- Expresiones
- Bloques de Código
- Funciones

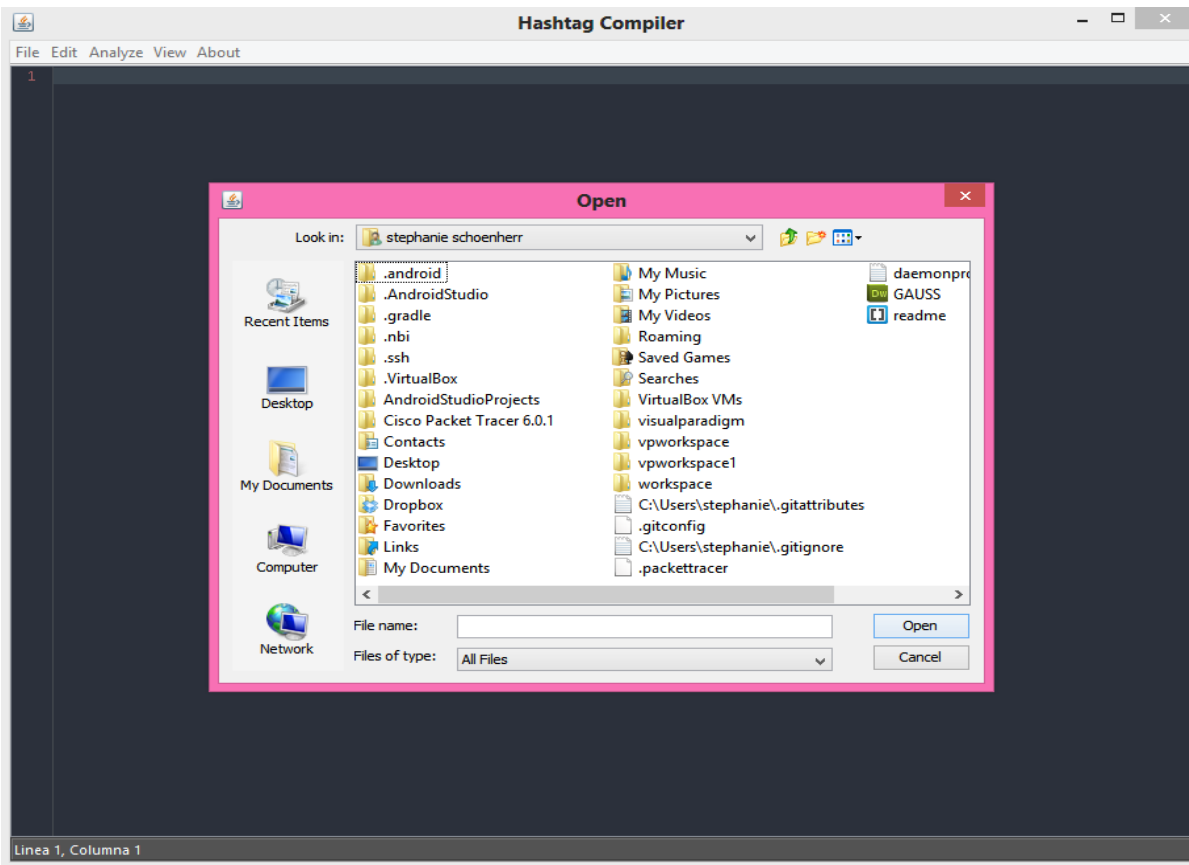
HASHTAG - COMPILER

Hashtag Compiler es el editor de texto pensado para escribir código de Hashtag. Este editor permite que el programa sea en formato de texto, y es capaz de colorear el código para facilitarnos la lectura y escritura. Cuenta con las opciones de gestión como abrir, guardar y editar archivos, así como ejecutar el código sin salir del editor y generar el árbol de sintaxis abstracto (AST) de un programa hecho en lenguaje Hashtag.

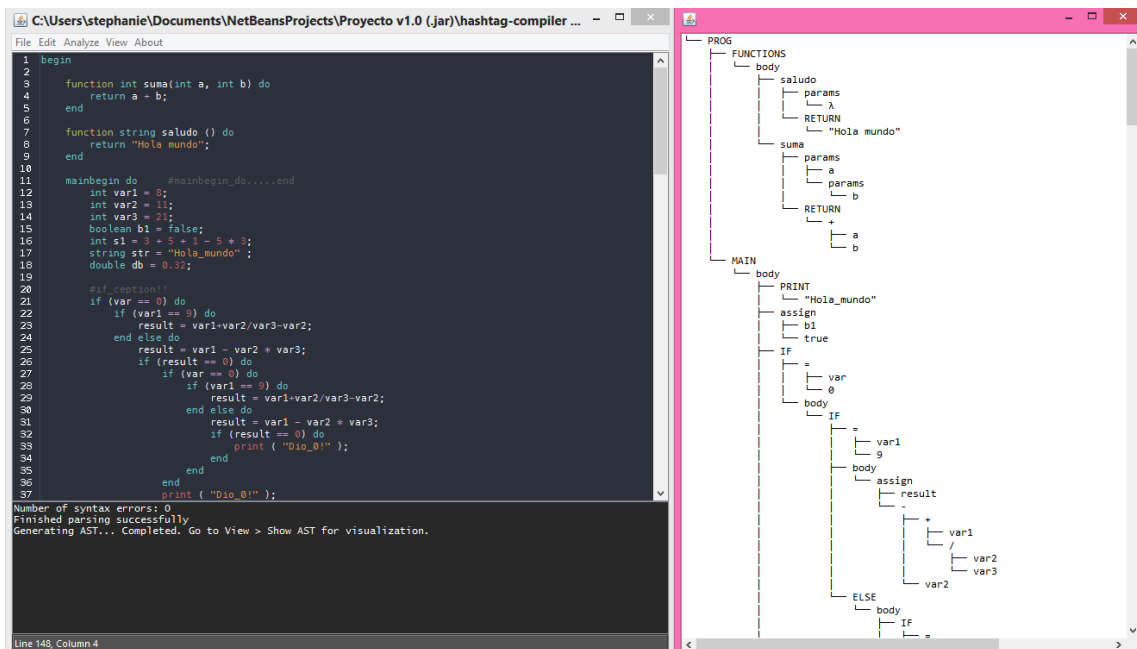
Pantalla Principal



Abrir Archivo



Ejemplo Programa con AST

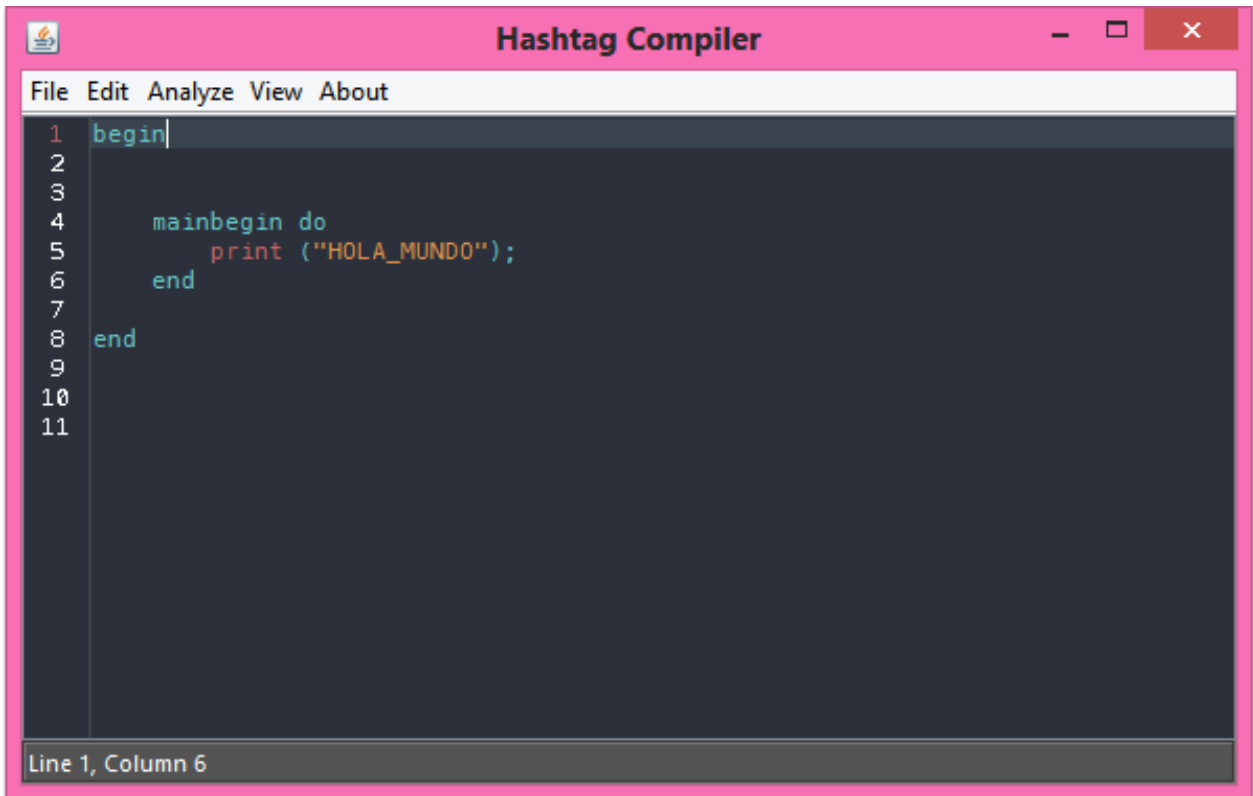


PROGRAMANDO EN HASHTAG

Para empezar a escribir un programa, primero necesitamos el entorno de trabajo "Hashtag Compiler", donde se creará un nuevo archivo de nuestro primer programa. Este debe comenzar con la palabra reservada "*begin*" y terminar con "*end*", dentro de ello se declarará el método principal o main que es el primero en ejecutarse al iniciar el programa, de la siguiente manera: *mainbegin do end* .

A continuación se podrá crear el programa deseado con declaración e inicialización de variables, funciones, llamados, estructuras de control, instrucciones y todo lo necesario para nuestro primer programa. Hashtag es sensible a mayúsculas y minúsculas, es muy importante saber que los nombres de las palabras reservadas todas deben ser minúsculas.

Primer Programa en Hashtag



The screenshot shows the Hashtag Compiler application window. The title bar is pink and contains the text "Hashtag Compiler" along with standard window controls. The menu bar includes "File", "Edit", "Analyze", "View", and "About". The main editing area has a dark background with light-colored text. The code is as follows:

```
1 begin|
2
3
4     mainbegin do
5         print ("HOLA_MUNDO");
6     end
7
8 end
9
10
11
```

The status bar at the bottom indicates "Line 1, Column 6".

IDENTIFICADORES

Un identificador es un contenedor de los datos que utiliza un programa. Cada nombre del identificador solo puede contener letras, números, alfanumérico, y el caracter guión bajo "_", este se crea a criterio del desarrollador siguiendo el patrón definido. El tipo de dato al que corresponde puede ser de tipo entero, punto flotante, caracter, cadena o booleano.

TIPOS DE DATOS

Hashtag dispone de 4 tipos de identificadores.

- Boolean: Tipos que almacena únicamente los valores *True* y *False*. EL resultado de la expresión lógica que aparece como condición en un bloque de decisión debe ser boolean.
- Int: Tipos que almacenan valores numéricos (Enteros).
- Double: Tipos que almacenan valores numéricos (Reales).
- Char: Tipos que almacena caracteres. Los valores de tipo carácter sirven para almacenar símbolos de escritura.
- String: Tipos que almacena un conjunto de caracteres.

DEFINICIÓN E INICIALIZACIÓN DE IDENTIFICADORES

Definición de un identificador

Antes de poder utilizar un identificador, ésta se debe declarar. Un identificador se especifica con el tipo y nombre. Si no se especifica el tipo del identificador creado, este muestra un error. El valor puede o no puede ser especificado al momento de la creación.

Tipo NombreIdentificador;

Tipo NombreIdentificador = Valor;

Tipo NombreIdentificador = Expresión;

Ejemplos de declaración e inicialización de variables:

```
int Var1;           {Declaración sin inicializar}
int Var2=10;        {Declaración inicializada con valor igual a 10}

double x;           {Declaración sin inicializar}
double y=2.5;       {Declaración inicializada con valor 2.5}

boolean b1=true;    {Declaración inicializada con valor true}
boolean b2=false;   {Declaración inicializada con valor false}

char caracter='a';   {Declaración de un caracter}

string cadena_Vacia=""; {Declaración de una cadena vacia}
string cadena="Hola mundo"; {Declaración de una cadena Hola mundo}
```

En el ejemplo se distingue las distintas formas que se puede nombrar un identificador, así como ver a que tipo pertenece y que valor contiene. Estos identificadores ahora podrán ser utilizados por el desarrollador en el caso que lo necesite.

Ejemplo #2

```
1
2  #Creacion Multiple Identificadores
3  int x, z, y;
4  double var1, var2;
5  string cad,mensaje,ss2;
6  #Tambien se pueden declarar un identificador e inicializarla con una expresión
7  int x = (a + b);
8
9  int numero = 8*10;
10
```

PALABRAS RESERVADAS

Estas palabras reservadas son las que emplea el lenguaje Hashtag y no pueden ser utilizadas como identificadores. Estas palabras reservadas tienen un significado específico para el compilador cada una tiene su función con el cual fue definido por la gramática del lenguaje. A continuación se muestra la lista de palabras reservadas correspondientes a Hashtag.

Lista palabras reservadas

boolean	if	Other	readint
break	int	End	readdouble
case	public	mainbegin	readchar
char	return	End	readstring
do	string	And	void
double	switch	Or	
else	true	Print	
false	while	Function	
for	not	Begin	

OPERADORES

Hashtag cuenta con:

- **Operadores aritméticos** Son operadores binarios que requieren siempre dos operandos la cual realizan operaciones aritméticas habituales: como suma (+), resta (-), multiplicación (*), división (/), y resto de la división (%).
- **Operador de asignación** el cual permite asignar un valor a una variable (=).
- **Operadores relacionales** sirven para realizar comparaciones de igualdad, desigualdad así como relación de menor o mayor. (<,>,<=,>=,!<,>=).
- **Operadores Lógicos** estos se utilizan para la construcción de expresiones lógicas (true, false, and , or , not).
- **Operadores Incremento** (++) incrementa en una unidad al identificador que se le aplica.
- **Operadores Decremento** (--) reduce en una unidad al identificador que se le aplica.
- **Operador de Asignación** (=) permite asignar valores a un identificador.

A continuación un ejemplo de cómo utilizar los operadores en Hashtag.

```
#Operadores Hashtag

double resultado, num1, num2=10;

resultado=num1*num2;
resultado=num1+num2;
resultado=num1-num2;
resultado=num1/num2;
resultado=num1%num2;

boolean mayorDeEdad, menorDeEdad;
    int edad = 21;
        mayorDeEdad = edad >= 18;    #mayorDeEdad será true
        menorDeEdad = not mayorDeEdad; #menorDeEdad será false

boolean fuego =true, agua=false, granizo=false;
boolean condicion= fuego or agua or graniz;
```

```

1
2
3  int x = 5;      #Identificadore de tipo entero inicializado con el valor igual a 5
4  int y = 5;
5  int z;
6  z = x++;       #operador de incremento, incrementa en una unidad.
7  z = Y--;       #operador de decremento, decrementa en una unidad.
8
9
10

```

COMENTARIOS

Hashtag cuenta con dos formas diferentes de introducir comentarios. Los comentarios son muy útiles para poder entender el código utilizado, facilitando de ese modo futuras revisiones y correcciones. Además permite que cualquier persona distinta al desarrollador original pueda comprender de manera más rápida el código. Realizar comentarios es una buena práctica que se debe aplicar.

Un comentario en una línea se puede colocar en cualquier parte del código. De la siguiente manera:

Este es un comentario en una línea.

Un comentario de múltiples líneas comienza con una llave izquierda seguido del texto y termina con una llave Derecha, se escribe de la siguiente manera:

{Comentario de multiples líneas

Ejemplo escrito en lenguaje Hashtag.}

FUNCIONES HASHTAG

Hashtag permite la creación de funciones que retornan un valor, dato o función, donde este realizará una tarea específica. Así como creación de los que no retornan ningún valor. Se debe especificar el tipo, nombre y la declaración de parámetros de la función. La lista de parámetros son identificadores separado por comas (,).

Ejemplo

```

function int suma (int a, int b) do

    return a + b;

end

function string saludo ( ) do

    return "Hola mundo";

end

```

LLAMADOS

A la hora de llamar una función, se escriben el nombre de la función y los parámetros entre paréntesis; pero en caso que no se indique los parámetros solo se incluye los paréntesis vacíos ().

Ejemplo:

```
Suma ( );  
Suma ( 5, 5);
```

PRINTS

La palabra reservada "print" seguido de los paréntesis donde se especifica la salida.

Ejemplo

<code>print ("Se_cambio_el_valor_de_var2");</code>	{Print de un mensaje}
<code>print ();</code>	{print vacío}
<code>print (a + b);</code>	{print expresión}
<code>print (Suma);</code>	{print función Suma}

ESTRUCTURAS DE CONTROL DEL FLUJO

Son los que permiten tomar decisiones y realizar un proceso repetidas veces. Estas estructuras son muy importantes ya que son las encargadas de controlar el flujo de ejecución de un programa.

IF

Estructura que permite ejecutar un conjunto de sentencias en función del valor que tenga la expresión de comparación. Por ejemplo:

Ejemplo #1

```
#IF ELSE
if(condicion) #Expresion Booleana
do
    #instrucciones que se ejecutan si la condicion es true
else
    #instrucciones que se ejecutan si la condicion es false
end

#IF ELSEIF ELSE
if (condicion1) #Expresion Booleana
do
    #instrucciones Hashtag1
else if(condicion2)
do
    #instrucciones Hashtag2
else if (condicion3)
do
    #instruccion Hashtag3
else
    #instruccion Hashtag4
end
end
end

#IF COMPARACION
end
if((diasemana >= 1) and (diasemana <=5))
do
    trabajar = false;
else do
    trabajar = true;
end
end
```

EJEMPLO#2

```
int usuario = 45;

if (usuario <= 18)
do
    print ("Usuario Menor");
else
    if(usuario > 45)
    do
        print ("Usuario Mayor");
    end
end
end
```

SWITCH

Esta es una alternativa de la estructura de control if else if else cuando se compara la misma expresión con distintos valores. En Hasktag se utilizan las palabras reservadas 'switch' y 'case': donde cada case corresponde con único valor de expresión. Para finalizar se utiliza la palabra reservada 'break'; indicando el fin de cada case.

```
3
4     switch(expresion) do
5         case 1:
6             Instruccion;
7             break;
8         case 2:
9             Instruccion;
10            break;
11        case 3:
12            Instruccion;
13            break;
14        other:
15    end
16    switch(expresion)
17        case 1:
18
19            print("Primer Caso");
20
21            break;
22        case 2:
23            print("Segundo Caso");
24            break;
25        case 3:
26            print("Tercer Caso");
27            break;
                other:
```

WHILE

La instrucción while permite crear bucles. Estos agrupan instrucciones las cuales se ejecutan continuamente hasta que una condición que se evalúa sea falsa.

Sintaxis While:

```
while (condicion)
do
    #sentencia que se ejecuta si la condicion es true
end
```

FOR

Es un bucle que se utiliza para ejecutar instrucciones controladas por un contador. Una herramienta muy útil que proporciona Hashtag. La sintaxis es la siguiente:

```
for (ExpresionIncial; Condicion; ExpresionEnCadaVuelta)
do
    #Instrucciones;
end
```

SENTENCIA RETURN

La sentencia return se emplea para salir de la secuencia de ejecución de un método y opcionalmente, devolver un valor. En caso de que una función devuelva alguna variable, este valor se deberá poner a continuación del return. Ejemplo : *Return value;*

DESARROLLO HASHTAG - COMPILER MEDIANTE JFLEX Y CUP

El desarrollo de nuestro compilador se utilizó las herramientas JFlex y CUP.

JFlex

JFlex es un metacompilador que permite generar rápidamente analizadores léxicos que se integran con Java, el cual toma una cadena como entrada de caracteres, y lo convierte en una secuencia de tokens.

Analizador Léxico

Nuestro analizador léxico, el cual es un archivo flex se divide en tres partes:

1. Código de usuario

El archivo comienza declarando el paquete en el cual está creado nuestro archivo léxico, el paquete se encuentra dentro de nuestro proyecto en java, una vez compilado generará un archivo java con el que trabajamos en nuestro programa.

2. Directivas Jflex

Dentro de las directivas, definimos nombres de estados, macros donde declaramos expresiones regulares que utilizamos para las reglas léxicas.

3. Reglas del Analizador

Esta sección contiene las acciones cuando se localiza un token que se ejecutaran cuando se encuentre una entrada válida para la expresión regular correspondiente.

YYINITIAL

Es el estado inicial del analizador léxico al escanear. Las expresiones regulares solo serán comparadas si se encuentra en ese estado inicial. Por lo cual se ignoran estados intermedios.

A continuación se muestra el código completo de JFlex

Código de Usuario

```
package hashtag;
import java_cup.runtime.Symbol;
%%

%class Lexer
%int
%unicode
%line
%column
%cup

%{
    /*para los simbolos generales*/
    private Symbol symbol(int type){
        return new JavaSymbol(type,yyline+1,yycolumn+1,yytext());
    }

    /*para el tipo de token con su valor*/
    private Symbol symbol (int type, Object value){
        return new JavaSymbol(type,yyline+1,yycolumn+1,yytext(),value);
    }

    StringBuilder string = new StringBuilder();
    StringBuilder comment = new StringBuilder();
%}

%eofval{
    return symbol(sym.EOF);|
%eofval}
```

Directivas de JFlex

```
/*-----MACROS-----*/
DIGITO=[0-9]+
LETRA=[a-zA-Z]
ALFANUMERICO={DIGITO}|{LETRA}
GUIONBAJO=[_]
ESPACIO=[" "]
SALTOLINEA=[\n\t\r]
IDENTIFICADOR={LETRA}({ALFANUMERICO}|{GUIONBAJO})*
NUMERO= {DIGITO}
REAL= {DIGITO} "." {DIGITO}
CARACTER= ' . '
LLAVEIZQ=[{]
LLAVEDER=[}]
COMILLAS=[""]
CONTENIDOCOMENT=( [^] ) *
HASHTAG=[#]
COMENTARIOUNALINEA={HASHTAG}.*

/*-----ESTADOS-----*/
%state COMMENT
%state SSTRING
%%
```

Reglas del Analizador

```
<YYINITIAL> {
    {COMILLAS}          {string.setLength(0); yybegin(SSTRING);}
    {LLAVEIZQ}          {comment.setLength(0); yybegin(COMMENT);}
    {COMENTARIOUNALINEA} {/*Ignore*/}
    {NUMERO}             {return symbol(sym.NUMERO, new Integer(yytext()));}
    {REAL}               {return symbol(sym.REAL, new Double(yytext()));}
    {CARACTER}           {return symbol(sym.CARACTER, new Character(yytext().charAt(1)));}
    {SALTOLINEA}         {/*Ignore*/}
    {ESPACIO}            {/*Ignore*/}

    /*-----OPERADORES-----*/
    "+"                 {return symbol(sym.SUMA);}
    "-"                 {return symbol(sym.MENOS);}
    "/"                 {return symbol(sym.DIV);}
    "*"                 {return symbol(sym.MULT);}
    ">"                 {return symbol(sym.MAYOR);}
    "<"                 {return symbol(sym.MENOR);}
    ">="               {return symbol(sym.MAYORIGUAL);}
    "<="               {return symbol(sym.MENORIGUAL);}
    "not"               {return symbol(sym.NOT);}
    "!="                {return symbol(sym.DIFERENTE);}
    "=="                {return symbol(sym.IGUAL);}
    "="                 {return symbol(sym.ASIGNACION);}

    /*-----SIGNOS-----*/
    "("                 {return symbol(sym.PARIZQ);}
    ")"                 {return symbol(sym.PARDER);}
    "%"                 {return symbol(sym.MOD);}
    ","                 {return symbol(sym.COMA);}
    ";"                 {return symbol(sym.PUNTOCOMA);}
    ":"                 {return symbol(sym.DOSPUNTOS);}

    /*-----TIPOS DE DATOS-----*/
    "int"               {return symbol(sym.INT);}
    "double"            {return symbol(sym.DOUBLE);}
    "char"              {return symbol (sym.CHAR);}
    "string"            {return symbol(sym.STRING);}
    "boolean"           {return symbol(sym.BOOLEAN);}
```

```

/*-----PALABRAS RESERVADAS-----*/
"and"          {return symbol(sym.AND);}
"or"           {return symbol(sym.OR);}
"for"          {return symbol(sym.FOR);}
"if"           {return symbol(sym.IF);}
"else"         {return symbol(sym.ELSE);}
"while"        {return symbol(sym.WHILE);}
"function"     {return symbol(sym.FUNCTION);}
"mainbegin"    {return symbol(sym.MAINBEGIN);}
"begin"        {return symbol(sym.BEGIN); }
"switch"       {return symbol(sym.SWITCH);}
"case"         {return symbol(sym.CASE);}
"do"           {return symbol(sym.DO);}
"end"          {return symbol(sym.END); }
"true"         {return symbol(sym.TRUE);}
"false"        {return symbol(sym.FALSE);}
"other"        {return symbol(sym.OTHER);}
"break"        {return symbol(sym.BREAK);}
"return"       {return symbol(sym.RETURN);}
"readint"      {return symbol(sym.READINT);}
"readdouble"   {return symbol(sym.READDOUBLE);}
"readstring"   {return symbol(sym.READSTRING);}
"readchar"     {return symbol(sym.READCHAR);}
"print"        {return symbol(sym.PRINT);}
"void"         {return symbol(sym.VOID);}
{IDENTIFICADOR} {return symbol(sym.IDENTIFICADOR, yytext());}
[^]            {int l = yyline+1; int c = yycolumn+1;
                GUI.console.setText(GUI.console.getText()+"Error: (line: " + l + ",
                + column: " + c + "). Unrecognized token " + yytext() + " : Lexical error\n");
                }
}

<COMMENT> {
    {CONTENIDOCOMENT} {comment.append(yytext());}
    {LLAVEDER}        {yybegin (YYINITIAL);}
}

<SSSTRING>{
    {COMILLAS}        {yybegin (YYINITIAL); return symbol(sym.CADENA, string.toString()); }
    .                  { string.append(yytext());}
}

```

CUP

CUP es un metacompilador utilizado para generar un analizador sintáctico ascendente con algoritmos LALR, el cual recibe de entrada un archivo con la estructura de la gramática y su salida es un parser escrito en Java listo para usarse.

Analizador Sintáctico

El analizador sintáctico nos permitirá comprobar que las expresiones utilizadas, ingresadas o leídas por el código generado de nuestro archivo flex y tenga la sintaxis correcta.

El código de nuestro archivo CUP se divide en tres partes importantes:

1. Código Parser

Este código se copia íntegramente a la clase final. Es aquí donde agregamos el manejo de errores.

2. Identificación de Errores

En esta sección de código es donde se manejan los errores encontrados, al interpretar lo que nos devuelve el método info de nuestro analizador sintáctico. Dentro de ese método se compara lo recibido en el método info.toString() y las constantes Symbol que son generadas por el archivo CUP dentro de la clase Sym.

3. Declaración de símbolos terminales, No terminales y creación de Gramática

Los terminales son los componentes léxicos (tokens) obtenidos por el analizador léxico Hashtag.flex.

No terminales son utilizados en la sección gramatical.

Sección Gramatical, se especifica la estructura del lenguaje mediante la gramática de nuestro analizador sintáctico.

El código completo de CUP es el siguiente:

Código Parser

```
package hashtag;
import java_cup.runtime.*;
import java.util.*;
import java.io.FileReader;

//-----PARSER CODE-----
parser code {:
    public ArrayList<Node> AST = new ArrayList();
    public int errors = 0;
    public void report_error(String message, Object info){
        StringBuilder m = new StringBuilder("");
        if(info instanceof java_cup.runtime.Symbol){
            m.append("Error: ");
            m.append(info);
            m.append(" : " + message);
        } else {
            if (info instanceof String) {
                errors++;
                m.append("      "+ errors + "==> " + info + " "+ message+"\n");
            }
        }

        GUI.console.setText(GUI.console.getText() + m.toString() + "\n");
    }

    public void report_fatal_error(String message, Object info){
        report_error(message, info);
        //System.exit(1);
    }
:}
```

Terminales y No terminales

```
//-----TERMINALES-----
terminal SUMA,MENOS,DIV,MULT,MAYOR,MENOR,MAYORIGUAL,MENORIGUAL,NOT,DIFERENTE,IGUAL,ASIGNACION;
terminal PARDER,PARIZQ,MOD,COMA,PUNTOCOMA,MAINBEGIN,DOSPUNTOS;
terminal AND,OR,FOR,IF,ELSE,WHILE,BEGIN,END,SWITCH,CASE,DO,FUNCTION,BREAK,RETURN,PRINT;
terminal INT,DOUBLE,CHAR,STRING,BOOLEAN,READSTRING,READCHAR,READINT,READDOUBLE,OTHER,VOID;
terminal Integer NUMERO;
terminal Double REAL;
terminal Character CARACTER;
terminal String CADENA, IDENTIFICADOR;
terminal Boolean FALSE, TRUE;

//-----NO-TERMINALES-----
non terminal Node Main,BodyList,BodyPart,Type,Asignacion,AsigValor,Program,FunctionList,FunctionPart,Stmts;
non terminal Node Instruccion,Llamado,ExpresionComparacion,OperadorComparacion,OperadorRelacional,ExpresionIncDec;
non terminal Node Incremento,Decremento;
non terminal Node Bloque,Loops,Conditional,ForLoop,WhileLoop,IfConditional,SwitchConditional,ExprFor,UnionExpresion;
non terminal Node Switch,ExprSwitchList,ExprSwitchPart;
non terminal Node ExpresionBooleana,ExpresionAritmetica,Factor,Term,ConditionGroup,Contenido,PrintParam;
non terminal Node Print,LlamadoMetodos,Booleana,Parametros,Multiple,Valor,EndFunction,Cadena,Return;
non terminal Node VarDeclare, VarInit, VarDeclarationList, VarDeclarationPart,Functions,Lectura,Lec;

//-----PRECEDENCIA-----

precedence left SUMA,MENOS;
precedence left MULT,DIV,MOD;
precedence left PARIZQ;
//leer usuario , llamado metodos
```

Fragmento del código de la Gramática

```
//-----GRAMATICA-----

start with Program;
Program ::= BEGIN Functions:f Main:mn END
        {:
          RESULT = new Node("PROG", f, mn);
          parser.AST.add(RESULT);
        :};
        |
        BEGIN Main:mn END
        {: RESULT = new Node("PROG", mn);
          parser.AST.add(RESULT); :}
        | BEGIN END error {: parser.report_error("syntax, missing token 'main' function.", "WRONG"); :} Stmts;

Functions ::= FunctionList:f1
          {:
            RESULT = new Node("FUNCTIONS", f1);
          :};

Main ::= MAINBEGIN Stmts:st
      {: RESULT = new Node("MAIN", st); :};

Stmts ::= DO BodyList:bd1 END
       {:
         RESULT = bd1;
       :};
       |
       DO Stmts:st END
       {: RESULT = st; :}
       |
       DO END
       {: RESULT = new Node("λ"); :}
       |
       DO error {: parser.report_error("block declaration. Possible unmatched 'do ... end'", "WRONG"); :} END
       ;

BodyList ::= BodyPart:bp BodyList:bd1
          {:
            RESULT = bd1.add(bp);
          :};
          |
          BodyPart:bp
          {: RESULT = new Node("body", bp); :}
          ;
```

```

BodyPart ::= Instruccion:ins PUNTOCOMA
          {: RESULT = ins; :}
          |
          Bloque:bq
          {: RESULT = bq; :}
          |
          Asignacion:asig PUNTOCOMA
          {: RESULT = asig; :}
          |
          error {: parser.report_error("start of expression.", "ILLEGAL"); :} BodyPart
          ;

/*DECLARACION VARIABLES ASIGNACION VARIABLES */

Instruccion ::= VarDeclare:vd
              {: RESULT = vd; :}
              |
              VarInit:vi
              {: RESULT = vi; :}
              |
              Llamado:l
              {: RESULT = l; :}
              |
              Lectura :l2
              {:RESULT = l2; :}
              |
              error {: parser.report_error("variable initialization, declaration or function call", "WRONG"); :} PUNTOCOMA
              ;

Lectura ::= IDENTIFICADOR ASIGNACION Lec;

Lec ::= READINT
      | READDOUBLE
      | READSTRING
      | READCHAR
      ;

VarDeclare ::= Type VarDeclarationList:vd1
             {:
               Node node = new Node(vd1.label);
               RESULT = new Node("declare",vd1.getChildren()).add(node);
             :};

VarDeclarationList ::= VarDeclarationPart:vdp COMA VarDeclarationList:vd1
                     {: RESULT = vd1.add(vdp); :}
                     |
                     VarDeclarationPart:vdp
                     {: RESULT = vdp; :};

VarDeclarationPart ::= IDENTIFICADOR:id
                     {: RESULT = new Node(id); :};

VarInit ::= VarDeclare:vd ASIGNACION AsigValor:av
           {: RESULT = new Node("assign",vd,av); :};

```

Valor	<pre> ::= NUMERO:num {: RESULT = new Node(Integer.toString(num)); :} IDENTIFICADOR:id {: RESULT = new Node(id); :} REAL:r {: RESULT = new Node(Double.toString(r)); :} ; </pre>
AsigValor	<pre> ::= Cadena:c {: RESULT = c; :} CARACTER:ch {: RESULT = new Node("'" + Character.toString(ch) + "'"); :} Booleana:bool {: RESULT = bool; :} ; </pre>
Booleana	<pre> ::= TRUE {: RESULT = new Node("true"); :} FALSE {: RESULT = new Node("false"); :} ; </pre>
Cadena	<pre> ::= CADENA:cad {: RESULT = new Node("\'" + cad + "\'"); :} ExpresionAritmetica:ea {: RESULT = ea; :} ; </pre>
Asignacion	<pre> ::= IDENTIFICADOR:id ASIGNACION AsigValor:av {: RESULT = new Node("assign", new Node(id), av); :} ExpresionIncDec:update {: RESULT = update; :}; </pre>
Type	<pre> ::= INT DOUBLE CHAR BOOLEAN STRING ; </pre>

```

/*LLAMADOS*/
Llamado ::= Print:pr
        { : RESULT = pr; :}
        |
        LlamadoMetodos:lm
        { : RESULT = lm; :}
        ;

LlamadoMetodos ::= IDENTIFICADOR:id PARIZQ Contenido:cont PARDER
        { : RESULT = new Node("function_call",new Node(id),cont); :}
        |
        IDENTIFICADOR:id PARIZQ PARDER
        { : RESULT = new Node("function_call", new Node(id), new Node("params",new Node("λ"))); :};

Contenido ::= AsigValor:av COMA Contenido:cont
        { : RESULT = cont.add(av); :}
        |
        AsigValor:av
        { : RESULT = new Node("params",av); :};

/*PARAMETROS*/
Parametros ::= Type IDENTIFICADOR:id Multiple:mult
        { : RESULT = new Node("params", new Node(id), mult); :}
        |
        Type IDENTIFICADOR:id
        { : RESULT = new Node("params", new Node(id)); :}
        |
        { : RESULT = new Node("params", new Node("λ")); :}
        |
        error { : parser.report_error("function arguments.", "WRONG"); :}Stmts
        ;

Multiple ::= COMA Parametros:param
        { : RESULT = param; :}
        ;

Print ::= PRINT PARIZQ PrintParam:pp PARDER
        { : RESULT = new Node("PRINT", pp); :};

PrintParam ::= IDENTIFICADOR:id
        { : RESULT = new Node(id); :}
        |
        CADENA:cad
        { : RESULT = new Node("λ"+cad+"λ"); :}
        |
        LlamadoMetodos:lm
        ;

```

EJEMPLO PROGRAMA COMPLETO HASHTAG

```
1  begin
2
3      function int suma(int a, int b) do
4          return a + b;
5      end
6
7      function string saludo () do
8          return "Hola mundo";
9      end
10
11     mainbegin do      #mainbegin_do...end
12         int var1 = -8;
13         int var2 = 11;
14         int var3 = 21;
15         boolean b1 = false;
16         int s1 = 3 + 5 + 1 - 5 * 3;
17         string str = "Hola_mundo" ;
18         double db = 0.32;
19
20         #if_ception!!
21         if (var == 0) do
22             if (var1 == 9) do
23                 result = var1+var2/var3-var2;
24             end else do
25                 result = var1 - var2 * var3;
26                 if (result == 0) do
27                     if (var == 0) do
28                         if (var1 == 9) do
29                             result = var1+var2/var3-var2;
30                         end else do
31                             result = var1 - var2 * var3;
32                             if (result == 0) do
33                                 print ( "Dio_0!" );
34                             end
35                         end
36                     end
37                     print ( "Dio_0!" );
38                 end
39             end
40         end
41     end
```

```

42     print (suma(var1,var3));
43     print (str);
44
45     double mult = db * var2;
46     int a,b,c = 0;
47
48     contador = 0;
49
50     while (contador != 5) do
51
52         for (int i = 0; i < 100; i++) do
53             if (i % 7 == 0) do
54                 contador++;
55             end
56         end
57     end
58
59     switch (contador) do
60
61         case 1: do
62             print ( "Encontro_solo_1." );
63             break ; #break_en_todo_case
64         end #end_case
65
66         case 2: do
67             print ( "Encontro_2." );
68             break ;
69         end #end_case
70
71         case 3: do
72             if (var == 0) do
73                 if (var1 == 9) do
74                     result = var1+var2/var3-var2;
75                 end else do
76                     result = var1 - var2 * var3;
77                     if (result == 0) do
78                         if (var == 0) do
79                             if (var1 == 9) do
80                                 result = var1+var2/var3-var2;
81                             end else do

```

```

82 result = var1 - var2 * var3;
83 if (result == 0) do
84     print ( "Dio_0!" );
85 end
86 end
87 end
88 print ( "Dio_0!" );
89 end
90 end
91 end
92     print ( "Encontro_3." );
93     break ;
94 end
95
96 case 4: do
97     print ( "Encontro_4." );
98     break ;
99 end
100
101 other : do
102     while ( true ) do
103         if ( true ) do
104             end
105         end
106         break ;
107     end
108
109 end
110
111 if (var1 == var2) do
112
113     for (int i = 0; i < 10; i++) do
114         var2 = var1 + i;
115     end
116     print ( "Se_cambio_el_valor_de_var2" );
117
118 end
119
120 #-----comments-----
121 #real*entero=real

```

```

122      #simple_comment!
123
124      #-----comments-----
125
126
127      int result;
128      if (var == 0) do
129          if (var1 == 9) do
130              result = var1+var2/var3-var2;
131          end else do
132              result = var1 - var2 * var3;
133              if (result == 0) do
134                  print ( "Dio_0!" );
135              end
136          end
137      end
138  sn
139      b1 = true;
140      print ( "Hola_mundo" );
141  end
142  endmain
143 end

```