Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Laboratorio de Organización de Lenguajes y
Compiladores 1 Sección “A”
Ing. Mario Bautista
Aux. Francisco José Puac
Eriksson José Hernández López – 2927 19159 1415

MANUAL DE USUARIO

“JPR”

INTRODUCCIÓN

El proyecto presentado, permite realizar un intérprete que ejecute instrucciones de alto nivel definidas en el nuevo lenguaje exclusivo para la USAC, el cual será llamado “JPR”, que será utilizado para que los alumnos que estén cursando el primer curso de programación acorde a la reforma curricular puedan ver las funcionalidades principales y la aplicación de los fundamentos de programación.

OBJETIVOS

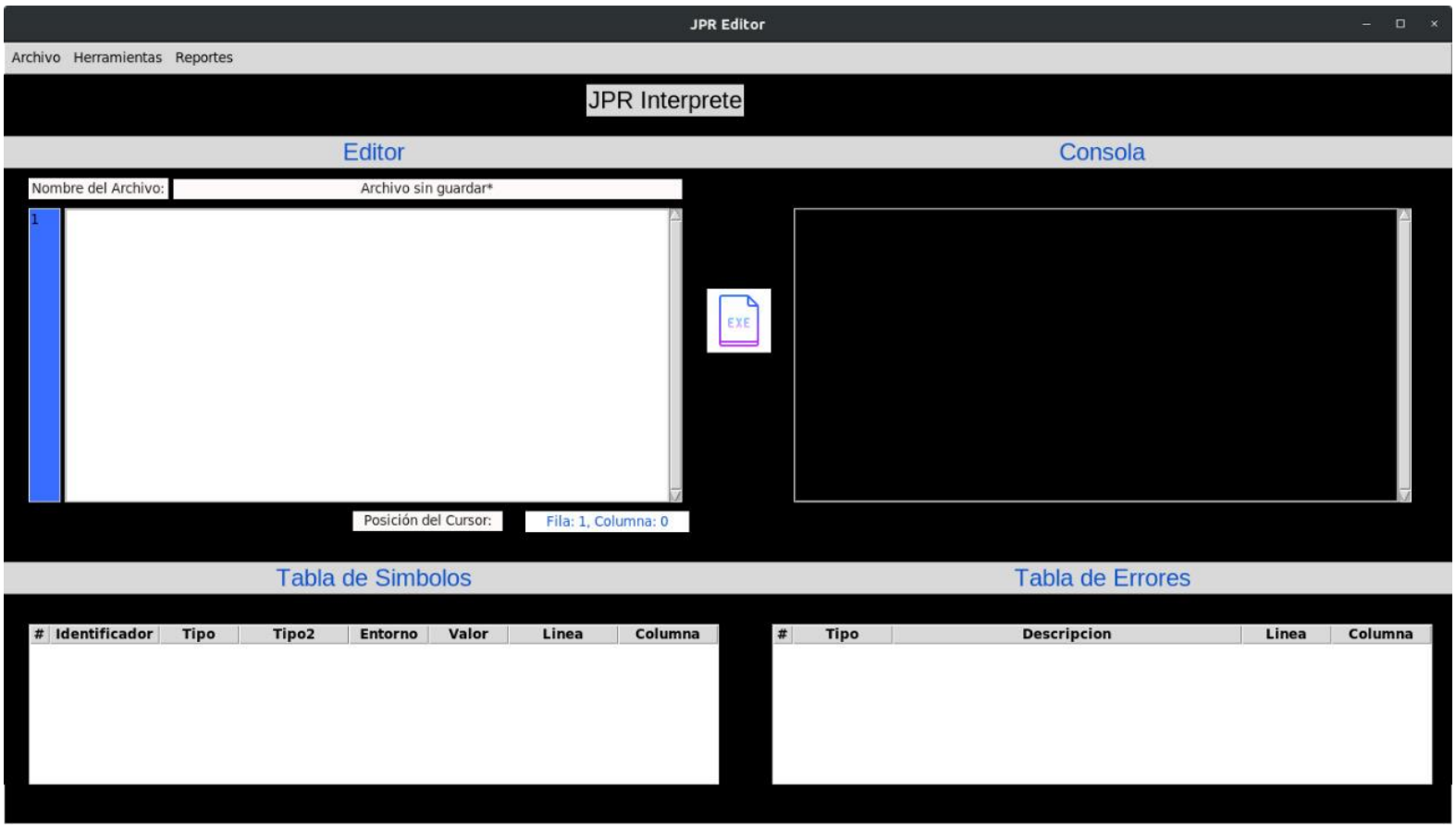
- **Objetivo General:**
  Aplicar los conocimientos sobre la fase de análisis léxico, sintáctico y semántico de un compilador para la realización de un intérprete completo, con las funcionalidades principales para que sea funcional.

- **Objetivos Específicos:**
  - Reforzar los conocimientos de análisis léxico, sintáctico y semántico para lacreación de un lenguaje de programación.
  - Aplicar los conceptos de compiladores para implementar el proceso deinterpretación de código de alto nivel.
  - Aplicar los conceptos de compiladores para analizar un lenguaje deprogramación y producir las salidas esperadas.
  - Aplicar la teoría de compiladores para la creación de soluciones de software.

1. Entorno de Trabajo

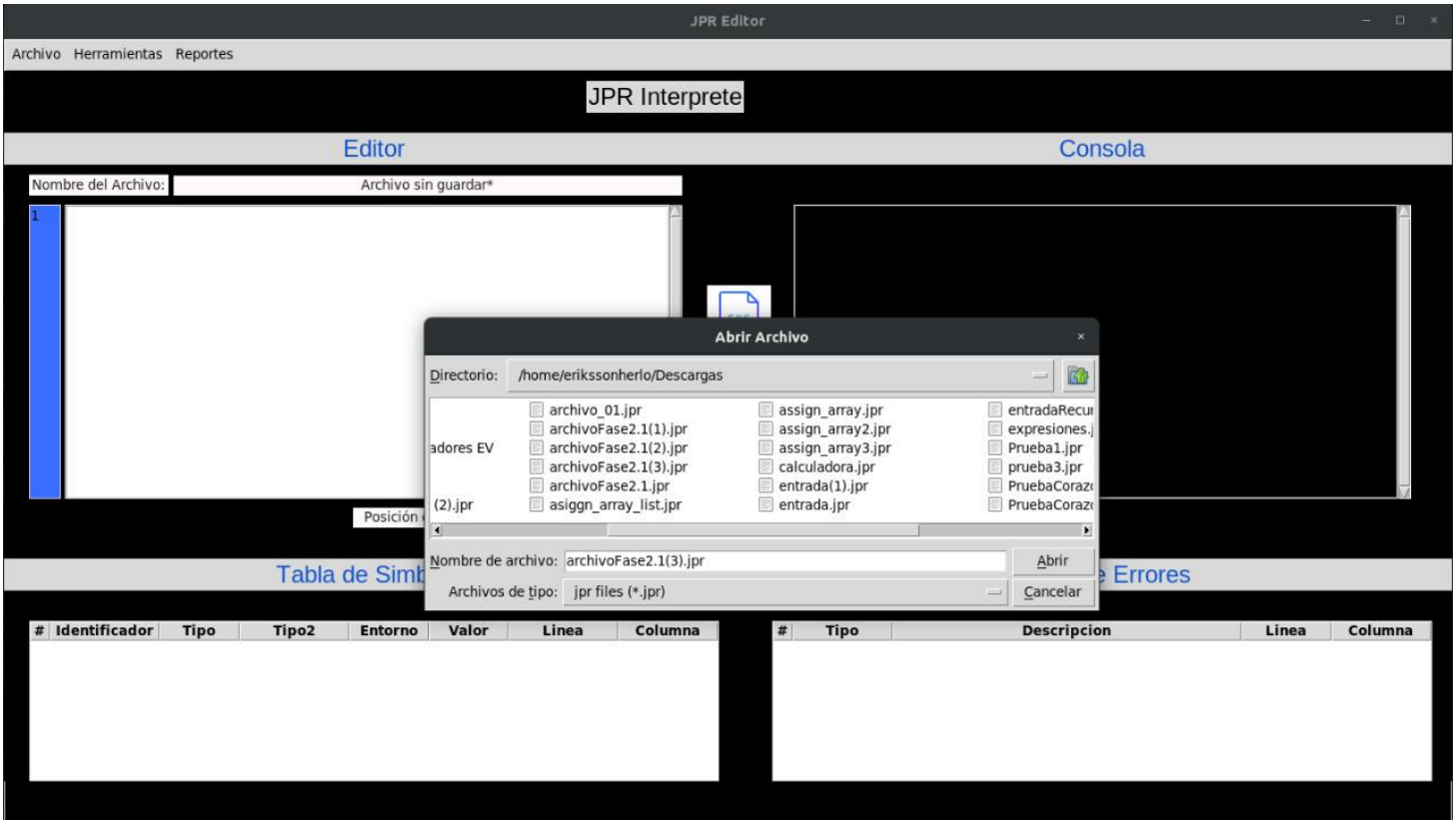
1.1. Editor

El editor será parte del entorno de trabajo, cuya finalidad será proporcionar ciertas funcionalidades, características y herramientas que serán de utilidad al usuario. La función principal del editor será el ingreso del código fuente que será analizado. En este se podrán abrir archivos .jpr y deberá mostrar la línea actual. Se recomienda queel editor de texto se realice con la herramienta Tkinter de Python para mayor compatibilidad. Queda a discreción del estudiante el diseño y la comunicación de este.



1.2. Funcionalidades

- **Crear archivos:** El editor deberá ser capaz de crear archivos en blanco.
- **Abrir archivos:** El editor deberá abrir archivos .jpr
- **Guardar**: El editor deberá guardar el estado del archivo en el que se estará trabajando.
- **Guardar Como**: El editor deberá guardar el estado del archivo en el que se estará trabajando con un nuevo nombre a elegir.



1.3. Herramientas

1.3.1. **Interpretar:** hará el llamado al intérprete, el cual se hará cargo de realizar los análisis léxico, sintáctico y semántico, además de ejecutar todas las sentencias.

1.3.2. **Debugger**: Característica que nos ayudará a ver el flujo de nuestro código al momento de ser ejecutado.

1.4. Reportes

1.4.1. **Reporte de Errores:** Se mostrarán todos los errores encontrados al realizar el análisis léxico, sintáctico y semántico.

1.4.2. **Generar Árbol AST (Árbol de Análisis Sintáctico):** se debe generar una imagendel árbol de análisis sintáctico que se genera al realizar los análisis.

1.4.3. **Reporte de Tabla de Símbolos:** Se mostrarán todas las variables, métodos y funciones que han sido declarados dentro del flujo del programa.

1.5. Carácterísticas

1.5.1. **Contador de Líneas**: El editor deberá poder visualizar las líneas en donde se encuentra el código, para mejorar la búsqueda de errores.

1.5.2. **Posición del Cursor**: El editor deberá poder visualizar el cursor en donde se encuentre activo en la entrada de código, para mejorar la búsqueda de errores.

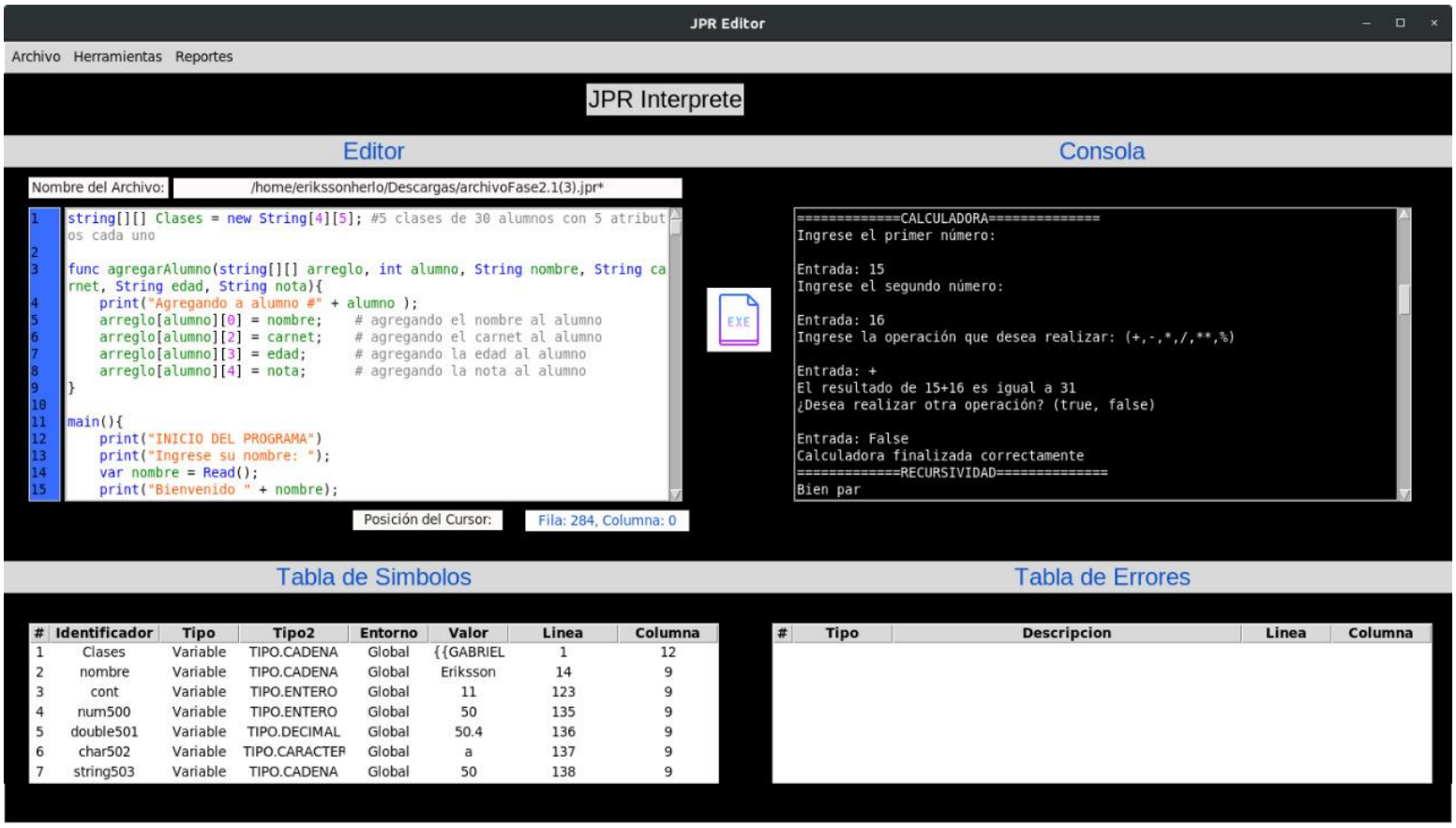
1.5.3. Se debe de pintar los tokens del código de entrada, el color de cada token sedefine en la siguiente tabla:

Token	Color
Palabras reservadas	Azul
Cadenas, caracteres	Naranja
Números	Morado
Comentarios	Gris
Otro	Negro

* Los reportes se deberán visualizar o abrir desde la aplicación, no desde un explorador de archivos.

1.6. Área de consola

En esta área se mostrarán los resultados, entradas de texto, mensajes y todo lo que sea indicado dentro del lenguaje.



#	Identificador	Tipo	Tipo2	Entorno	Valor	Linea	Columna
1	Clases	Variable	TIPO.CADENA	Global	{{GABRIEL	1	12
2	nombre	Variable	TIPO.CADENA	Global	Eriksson	14	9
3	cont	Variable	TIPO.ENTERO	Global	11	123	9
4	num500	Variable	TIPO.ENTERO	Global	50	135	9
5	double501	Variable	TIPO.DECIMAL	Global	50.4	136	9
6	char502	Variable	TIPO.CARACTER	Global	a	137	9
7	string503	Variable	TIPO.CADENA	Global	50	138	9

#	Tipo	Descripcion	Linea	Columna

2. Descripción del Lenguaje

2.1. Declaración y asignación de variables

Una variable deberá de ser declarada antes de poder ser utilizada. Todas las variables tendrán un tipo de dato y un nombre de identificador. Las variables podrán ser declaradas global y localmente. La declaración de variables debe de tener la palabra “var”, seguido de un identificador y su expresión. Cuando se quiera declarar o asignar una variable que tenga un arreglo, esta será tomada por **referencia**.

Las variables solamente declaradas tendrán el valor null por defecto, al asignarse otro valor, las variables no podrán cambiar de tipo de dato, **solamente si se asigna nuevamente un valor null**.

2.2. Casteos

Los casteos son una forma de indicar al lenguaje que convierta un tipo de dato en otro, por lo que, si queremos cambiar un valor a otro tipo, es la forma adecuada de hacerlo. Para hacer esto, se colocará la palabra reservada del tipo de dato destino entre paréntesis seguido de una expresión.

El lenguaje aceptará los siguientes casteos:

- **Int a double**
- **Double a Int**
- **Int a String**
- **Int a Char**
- **Double a String**
- **Char a int**
- **Char a double**
- **String a Int**
- **String double**

2.3. Incremento y Decremento

Los incrementos y decrementos nos ayudan a realizar la suma o resta continua de una variable de uno en uno, es decir si incrementamos una variable, se incrementará de uno en uno, mientras que, si realizamos un decremento, hará la operación contraria. Solamente realizará operaciones con tipo de dato Int y Double.

2.4. Arreglos

Los arreglos son una estructura de datos de tamaño fijo que pueden almacenar valores de forma limitada, y los valores que pueden almacenar son de un único tipo; int, double, boolean, char o string. **El lenguaje permite el uso de arreglos de múltiples dimensiones.**

Observaciones:

- La posición de cada vector será N-1. Por ejemplo, si deseo acceder al primer valorde un arreglo debo acceder como val[0].

2.4.1. Declaración de Arreglos

Al momento de declarar un arreglo, tenemos dos tipos que son:

- **Declaración tipo 1:** En esta declaración, se indica por medio de una expresión numérica del tamaño y la dimensión que se desea el arreglo, tomando el valor null para cada espacio. Sin embargo, estos valores no podrán cambiar de tipo.
- **Declaración tipo 2:** En esta declaración, se indica por medio de una listade valores separados por coma, los valores que tendrá el arreglo, en estecaso el tamaño del arreglo será el de la misma cantidad de valores de la lista. Si es de más dimensiones el arreglo, la lista deberá de contener las listas necesarias para la asignación de todos los elementos del arreglo.

2.4.2. Acceso a Arreglos

Para acceder al valor de una posición de un arreglo, se colocará el nombre del arreglo seguido de '[' EXPRESION ']'.

2.4.3. Modificación de Arreglos

Para modificar el valor de una posición de un arreglo, se debe colocar el nombre del arreglo seguido de '[' EXPRESION ']' = EXPRESION

2.5. Sentencias de control

Estas sentencias modifican el flujo del programa introduciendo condicionales. Las sentencias de control para el programa son el IF y el SWITCH.

OBSERVACIONES:

- También, entre las sentencias pueden tener Ifs anidados.

2.5.1. if

La sentencia if ejecuta las instrucciones sólo si se cumple una condición. Si la condición es falsa, se omiten las sentencias dentro del if. Else, else if, y combinaciones

2.5.2. Switch Case

Switch case es una estructura utilizada para agilizar la toma de decisiones múltiples, trabaja de la misma manera que lo harían sucesivos else-if.

4.17.2.1. Switch

Estructura principal del switch, donde se indica la expresión a evaluar.

4.17.2.2. Case

Estructura que contiene las diversas opciones a evaluar con la expresión establecida en el switch.

4.17.2.3. Default

Estructura que contiene las sentencias si en dado caso no haya salido del switch por medio de una sentencia **break**.

OBSERVACIONES:

- Si la cláusula "case" no posee ninguna sentencia "break", al terminar todas las sentencias del case ingresado, el lenguaje seguirá evaluando las demás opciones.

2.6. Sentencias cíclicas

Los ciclos o bucles, son una secuencia de instrucciones de código que se ejecutan una vez tras otra mientras la condición, que se ha asignado para que pueda ejecutarse, sea verdadera. En el lenguaje actual, se podrán realizar 2 sentencias cíclicas que se describen a continuación.

OBSERVACIONES:

- Es importante destacar que pueden tener ciclos anidados entre las sentencias a ejecutar.
- También, entre las sentencias pueden tener ciclos diferentes anidados.

2.6.1. While

El ciclo o bucle While, es una sentencia que ejecuta una secuencia de instrucciones mientras la condición de ejecución se mantenga verdadera.

2.6.2. For

El ciclo o bucle for, es una sentencia que nos permite ejecutar N cantidad de veces la secuencia de instrucciones que se encuentra dentro de ella.

OBSERVACIONES:

- Para la actualización de la variable del ciclo for, se puede utilizar:
  - **Incremento | Decremento:** i++ | i--
  - **Asignación:** como i=i+1, i=i-1, i=5, i=x, etc, es decir cualquier tipo de asignación.
- Dentro pueden venir N instrucciones

2.7. Sentencias de transferencia

Las sentencias de transferencia nos permiten manipular el comportamiento de los bucles, ya sea para detenerlo o para saltarse algunas iteraciones. El lenguaje soporta las siguientes sentencias:

4.17.1. Break

La sentencia break hace que se salga del ciclo inmediatamente, es decir que el código que se encuentre después del break en la misma iteración no se ejecutara y este se saldrá del ciclo.

4.17.2. Continue

La sentencia continue puede detener la ejecución de la iteración actual y saltar a la siguiente. La sentencia continue siempre debe de estar dentro de un ciclo, de lo contrario será un error.

4.17.3. Return

La sentencia return finaliza la ejecución de un método o función y puede especificar un valor para ser devuelto a quien llama a la función. Puede ser devuelto cualquier dato primitivo, null o un arreglo.

2.8. Funciones

Una función es una subrutina de código que se identifica con un nombre, un conjunto de parámetros y de instrucciones. Para este lenguaje las funciones serán declaradas indicando que serán funciones, luego un identificador para la función, seguido de una lista de parámetros dentro de paréntesis (esta lista de parámetros puede estar vacía en el caso de que la función no tenga parámetros).

Cada parámetro debe estar compuesto por su tipo seguido de un identificador, para el caso de que sean varios parámetros se debe utilizar comas para separar cada parámetro y en el caso de que no se usen parámetros no se deberá incluir nada dentro de los paréntesis. Luego de definir la función y sus parámetros se declara el cuerpo de la función, el cual es

un conjunto de instrucciones delimitadas por llaves {}.

Como observación, se podrán ingresar arreglos como parámetros en las funciones, donde se indica su tipo seguido de la nomenclatura de un arreglo vacío (arreglo[]) dependiendo en el número de dimensiones que tenga el arreglo.
La función puede llevar o no un valor de retorno, si no posee alguna sentencia return y no devuelve ningún valor, será considerado como un método.

Cabe a destacar que **no habrá sobrecarga de funciones y métodos** dentro de este lenguaje por lo que solo puede existir una función o método con el id declarado por lo que si se crea otra función o método con un id previamente utilizado esto debe de generar un error de tipo semántico.

2.9. Llamadas

Una llamada a una función cumple con la tarea de ejecutar el código de una función del código de entrada ingresado.
La llamada a una función puede devolver un resultado que ha de ser recogido, bien asignándolo a una variable del tipo adecuado o bien integrándolo en una expresión.

La sintaxis de las llamadas de las funciones es la siguiente:

OBSERVACIONES:

- Al momento de ejecutar cualquier llamada, no se diferenciarán entre métodos y funciones, por lo tanto, podrá venir una función que retorne un valor como un método, pero la expresión retornada no se asignará a ninguna variable.
- Se podrán llamar métodos y funciones antes que se encuentren declaradas, paraello se recomienda realizar 2 pasadas del AST generado: La primera paraalmacenar todas las funciones, y la segunda para las variables globales y la función main.
- Se pueden enviar arreglos como parámetros en la llamada.

2.10. Función Print

Esta función nos permite imprimir expresiones con valores únicamente de tipo int, double, booleano, string y char.

2.11. Función Read

Esta función nos permite obtener valores que queramos ingresar en el momento de ejecución del código, **el valor ingresado se tomará como un string**, por lo que, si se quiereingresar un número, para tomarlo como tal se debe de castearlo.

Se recomienda utilizar una pausa, poder escribir el valor requerido en el área de consola, y almomento de presionar “ENTER”, que el programa capture el valor de la última línea de la consola y seguir con su ejecución.

2.12. Función toLower

Esta función recibe como parámetro una expresión de tipo **String** y retorna una nuevacadena con todas las letras minúsculas.

2.13. Función toUpper

Esta función recibe como parámetro una expresión de tipo cadena retorna una nuevacadena con todas las letras mayúsculas.

4.24. Funciones nativas

4.24.1 Length

Esta función recibe como parámetro un arreglo o una cadena y devuelve el tamaño de este.

Nota: Si recibe como parámetro un tipo de dato no especificado, se considera un error semántico.

4.24.2 Truncate

Esta función recibe como parámetro un valor numérico. Permite eliminar los decimales de unnúmero, retornando un entero.

Nota: Si la función recibe un valor no numérico, se considera un error semántico.

4.24.3 Round

Esta función recibe como parámetro un valor numérico. Permite redondear los númerosdecimales según las siguientes reglas:

Si el decimal es mayor o igual que 0.5, se aproxima al número superiorSi el decimal es menor que 0.5, se aproxima al número inferior.

4.24.4 Typeof

Esta función retorna una cadena con el nombre del tipo de dato evaluado.

4.25 Main

Para poder ejecutar todo el código generado dentro del lenguaje, se utilizará la sentenciaMAIN para indicar en dónde se iniciará a ejecutar el programa.

Observaciones:

- Puede venir solo una vez, si viene más de una vez deberá lanzar error y no podráejecutar ninguna instrucción**.**

3. Reportes

Los reportes son una parte fundamental de JPR, ya que muestra de forma visual las herramientasutilizadas para realizar la ejecución del código.

A continuación, se muestran ejemplos de estos reportes. (Queda a discreción del estudiante eldiseño de estos, solo se pide que sean totalmente legibles).

3.1. Tabla de Símbolos

Este reporte mostrará la tabla de símbolos después de la ejecución. Se deberán mostrar todaslas variables, funciones y métodos declarados, así como su tipo y toda la información que se considere necesaria.

Tabla de Simbolos

#	Identificador	Tipo	Tipo2	Entorno	Valor	Linea	Columna
1	Clases	Variable	TIPO.CADENA	Global	{{GABRIEL	1	12
2	nombre	Variable	TIPO.CADENA	Global	Eriksson	14	9
3	cont	Variable	TIPO.ENTERO	Global	11	123	9
4	num500	Variable	TIPO.ENTERO	Global	50	135	9
5	double501	Variable	TIPO.DECIMAL	Global	50.4	136	9
6	char502	Variable	TIPO.CARACTER	Global	a	137	9
7	string503	Variable	TIPO.CADENA	Global	50	138	9

3.2. Tabla de Errores

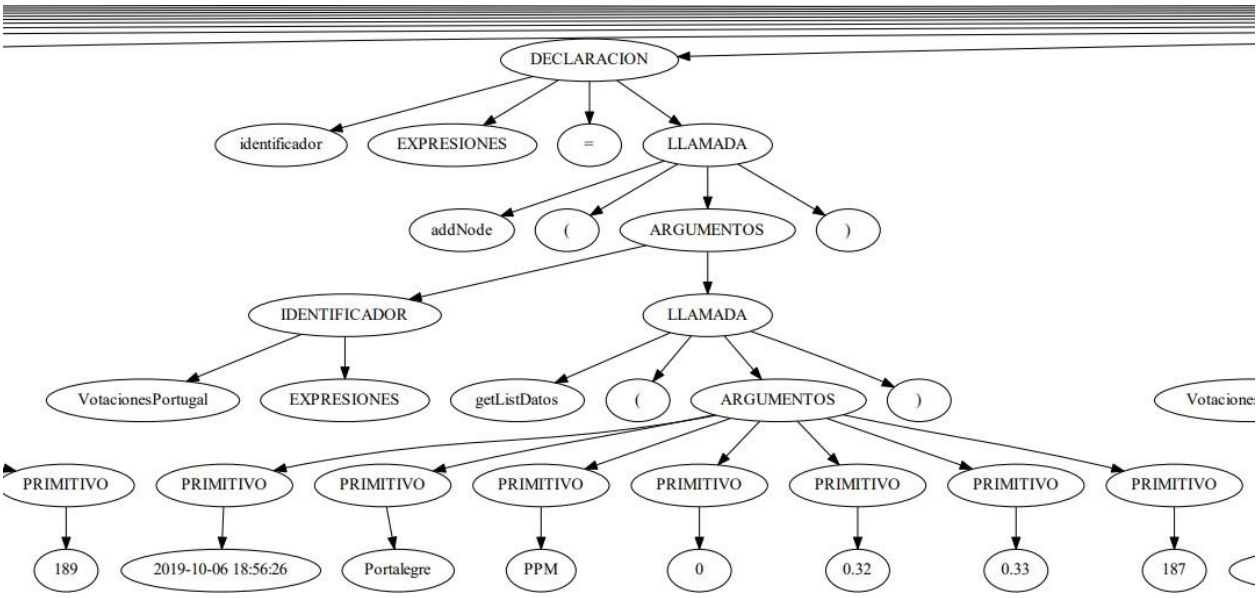
El reporte de errores debe contener la información suficiente para detectar y corregir errores enel código fuente.

Tabla de Errores

#	Tipo	Descripcion	Linea	Columna
1	Semantico	Tipo de Dato diferente en Asignacion	9	2
2	Semantico	No se puede castear para Int.	10	9

3.3. AST

Este reporte muestra el árbol de sintaxis producido al analizar los archivos de entrada. Este debe de representarse como un grafo. Se deben mostrar los nodos que el estudiante considerenecesarios para describir el flujo realizado para analizar e interpretar sus archivos de entrada.



3.4. Salidas en Consola

La consola es el área de salida del intérprete. Por medio de esta herramienta se podrán visualizar las salidas generadas por la función nativa “print”, así como los errores léxicos,sintácticos y semánticos.