MANUAL TÉCNICO

Ruth Nohemy Ardón Lechuga
Carnet: 201602975

Descripción de aplicación

Augus es un lenguaje de programación, basado en PHP y en MIPS. Su principal funcionalidad es ser un lenguaje intermedio, ni de alto nivel como PHP ni de bajo nivel como el lenguaje ensamblador de MIPS.

El lenguaje tiene dos restricciones: la primera, es que cada instrucción es una operación simple; y la segunda, es que en cada instrucción hay un máximo de dos operandos y su asignación.

Es un lenguaje débilmente tipado, sin embargo, si se reconocen cuatro tipos de datos no explícitos: entero, punto flotante, cadena de caracteres y arreglo. Para manejar el flujo de control se proporciona la declaración de etiquetas.

Requerimientos mínimos

Definición de los requerimientos técnicos del sistema: Procesador I3, espacio disponible de al menos 5 MB, sistema operativo de Windows 10.

Aplicaciones/Librerías necesarias

- Visual Studio code (O cualquier editor de texto)
- Librería PLY
- Graphviz 2.28
- Librería Magicsticklibs
- Módulo tkinter
- Módulo easygui

Diagrama de clases

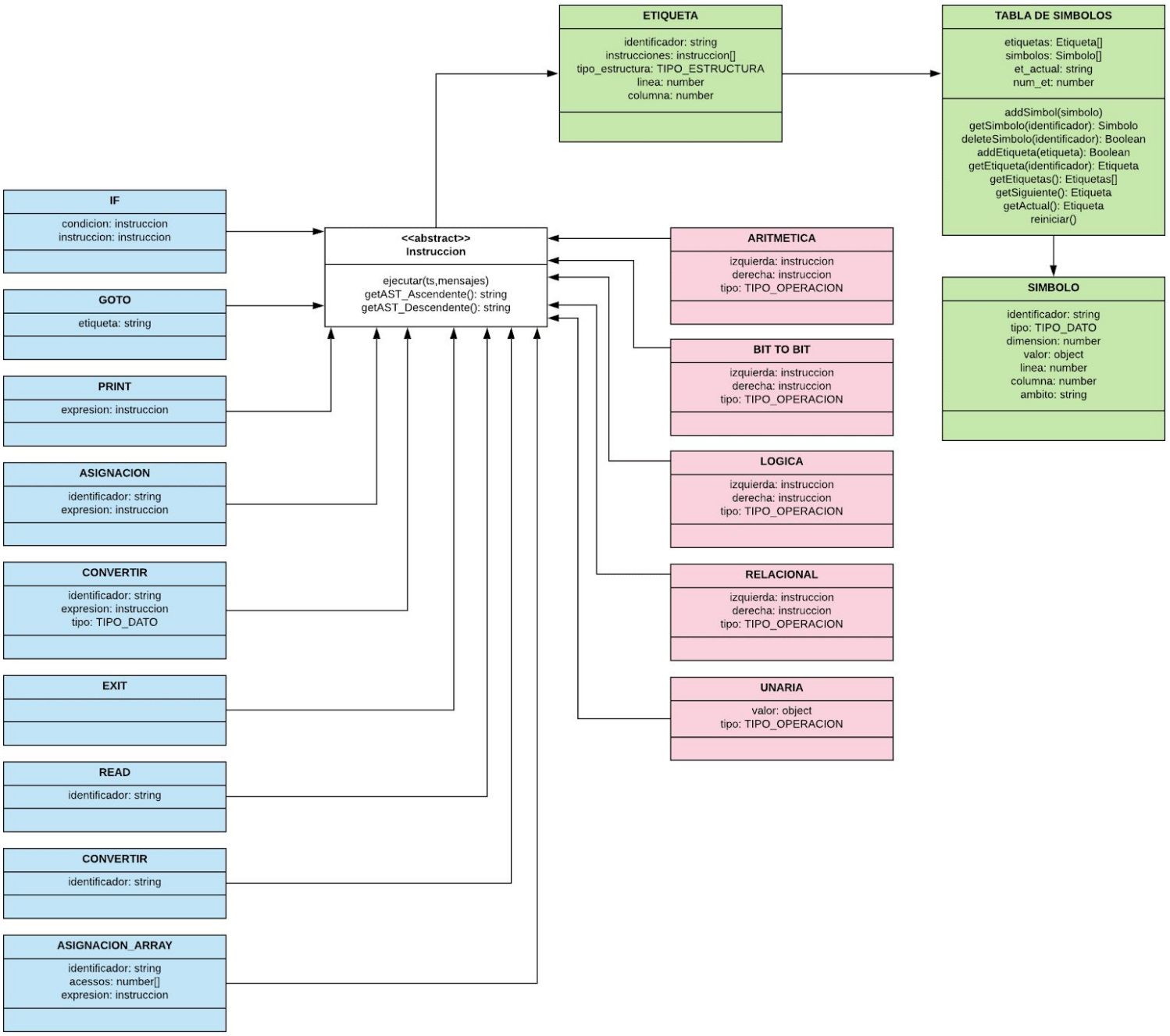
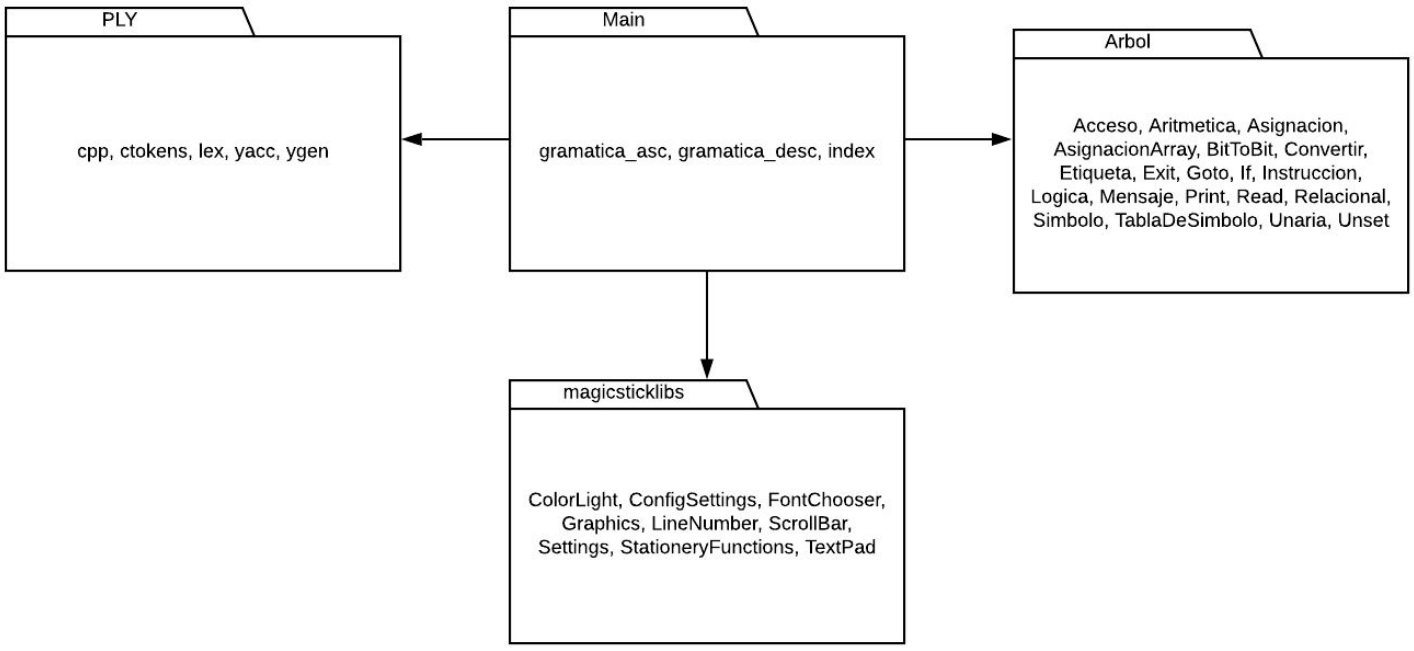


Diagrama de paquetes



Descripción de herramientas utilizadas

- PLY: Generador de Python de analizadores léxicos y sintácticos.

- Tkinter: Estándar para la interfaz gráfica de usuario para Python.

- Magicsticklibs: Editor simple en Python, repositorio público en: https://github.com/surajsinghbisht054/MagicStick_Editor. Se extrajeron las clases: ColorLight, ConfigSetting, FontChooser, Graphics, LineNumber, ScrollBar, Settings, StationeryFunctions y TextPad para poder realizar la interfaz gráfica.

- EasyGUI: Librería para producir mensajes y alertas.

- Graphviz: conjunto de herramientas de software para el diseño de diagramas definido en el lenguaje descriptivo DOT.

Descripción de clases y métodos

- gramatica_asc: Contiene el analizador léxico y sintáctico ascendente, este análisis se realiza mediante atributos sintetizados. Se lleva una variable global para los errores léxicos y sintácticos. Además un arreglo para producir el reporte gramatical mientras se realiza el análisis.

- gramatica_desc: Contiene el analizador léxico y sintáctico ascendente, este análisis se realiza mediante atributos sintetizados. Se lleva una variable global para los errores léxicos y sintácticos. Además un arreglo para producir el reporte gramatical mientras se realiza el análisis.

- index: Clase en la cual empieza la ejecución, contiene toda la interfaz gráfica del IDE y los métodos de acceso para la ejecución correcta de los archivos de entrada.

- Instrucción: Interfaz con dos métodos: ejecutar y getArbol. Todas las clases que implementen esta interfaz, deberán de implementar dichos métodos, todas esas clases se considerarán de tipo interfaz.
    - Ejecutar: Método que se encargará de realizar las acciones correspondientes a la clase que se quiere ejecutar, recibe como parámetro la tabla de símbolos y la lista de mensajes.
    - getAST_Ascendente: Método que construirá el árbol ascendente de análisis sintáctico de la clase.
    - getAST_Descendente: Método que construirá el árbol ascendente de análisis sintáctico de la clase.

- Símbolo: Clase que contiene todos los atributos esenciales y característicos de un símbolo en un compilador.

- Etiqueta: Clase que contiene todos los atributos de una etiqueta, es decir el identificador, todas las instrucciones que la constituyen y el tipo de estructura que representa.

- Tabla de símbolos: Estructura donde se almacenarán todos los símbolos que se encuentren en la ejecución del programa.
    - getSimbolo: Recibe como parámetro la cadena de identificador del símbolo y retorna símbolo en caso exista.
    - addSimbolo: Recibe como parámetro el nuevo signo y lo inserta en el diccionario.
    - deleteSimbolo: Borra un símbolo de la tabla de símbolos.
    - addEtiqueta: Agrega una etiqueta a la tabla de símbolos.
    - getEtiqueta: Recibe como parámetro el identificador de la etiqueta, retorna dicha etiqueta y la establece como el entorno actual.
    - getEtiquetas: Retorna todas las etiquetas existentes.
    - getSiguiente: Retorna la siguiente etiqueta de la etiqueta actual, establece esta como el entorno actual.
    - getEtActual: Retorna la etiqueta actual.
    - reiniciar: Vacía las etiquetas y símbolos existentes en la tabla de símbolos.

Explicación de acciones semánticas

init → labels
//se sintetiza labels a init

labels → labels label
//se agrega label a la lista actual de labels, se sintetiza el resultado

labels →label
//se sintetiza el resultado label al resultado

label → exp_label : instrucciones
//se crea la etiqueta

instrucciones → instrucciones instruccion
//se agrega instrucción a la lista actual de instrucciones, se sintetiza el resultado

instrucciones → instruccion
//se sintetiza el instruccion al resultado

instruccion → print_inst
| goto_inst
| exit_inst
| unset_inst
| if_inst
| asig_inst
| asig_array_inst
//se sintetiza la instruccion al resultado

print_ins → PRINT (expresion_simple) ;
//se crea la instrucción print

goto_inst → GOTO exp_label ;
//se crea la instrucción goto

exit_inst → EXIT ;
//se crea la instrucción exit

unset_inst → UNSET (asignable) ;
//se crea la instrucción unset

if_inst → IF (expresion) instruccion
//se crea la instrucción if

asig_inst → asignable = expresion ;
//se crea la instrucción asignación

asig_inst → asignable = (INT) expresion_simple ;
| asignable = (FLOAT) expresion_simple ;
| asignable = (CHAR) expresion_simple ;
//se crea la instrucción de conversión en torno al tipo de datos especificado

asig_inst → asignable = read ( ) ;
//se crea la instrucción read

asig_inst → asignable = ARRAY ( ) ;
//se crea la instrucción de asignación de un array

asig_array_inst → asignable accesos = expresion ;
//se crea la instrucción de asignación a arreglos

accesos → accesos acceso
//se agrega acceso a la lista actual de accesos, se sintetiza el resultado

accesos → acceso
//se sintetiza el acceso al resultado

acceso → [expresion_simple]
//se sintetiza el resultado de expresion_simple

asignable → temporal
| parametro
| retorno
| pila
| ra
| sp
//se sintetiza el resultado

expresion → expresion_simple + expresion simple
| expresion_simple - expresion_simple
| expresion_simple * expresion_simple
| expresion_simple / expresion_simple
| expresion_simple % expresion_simple
| expresion_simple && expresion_simple
| expresion_simple || expresion_simple
| expresion_simple XOR expresion_simple
| expresion_simple & expresion_simple
| expresion_simple | expresion_simple
| expresion_simple ^ expresion_simple
| expresion_simple << expresion_simple

| expresion_simple >> expresion_simple
| expresion_simple == expresion_simple
| expresion_simple != expresion_simple
| expresion_simple > expresion_simple
| expresion_simple < expresion_simple
| expresion_simple >= expresion_simple
| expresion_simple <= expresion_simple
| expresion_simple

//se crea la expresión en torno al tipo de operador que se especifique

expresion_simple → - expresion_simple
//se crea la expresión negativa

expresion_simple → ABS (expresion_simple)
//se crea la expresión absoluto

expresion_simple → ! expresion_simple
//se crea la expresión not

expresion_simple → ~ expresion_simple
//se crea la expresión not bit a bit
expresion_simple → asignable accesos
//se crea la expresión de acceso a arreglos

expresion_simple → temporal
| parametro
| retorno
| pila
| ra
| sp
| entero
| decimal
| cadena

//se sintetiza la expresión