GeoWall-E

Proyecto Programación III

Fabio Víctor Alonso Bañobre
C-111

GeoWall-E-2k3 es un programa que permite al usuario representar conceptos geométricos que conocemos de toda la vida como: puntos, rectas (líneas), semirrectas (rayos) o circunferencias. También permite comprobar relaciones entre estos conceptos concetos como por ejemplo: la intercepción. Para lograr este último objetivo contará con: un lienzo, una regla y un compás "cibernético"; permitiendo graficar tanto líneas como circunferencias.

PARTES FUNDAMENTALES DEL PROYECTO

1- Un compilador: encargado de realizar el análisis tanto Léxico como Sintáctico.
2- Una interfaz gráfica: visualiza el resultado de la compilación.

FASES DEL COMPILADOR

A gran escala el compilador se divide en tres fases.

1. Lectura de la expresión tecleada por el usuario: proceso en el cual el **LEXER** cumple su función de devolver una lista de Tokens (secuencia de caracteres que posee un valor y un tipo específico) una vez escaneado el código.

2. Creación de un Árbol de Sintaxis Abstracta: aquí es donde cobra vida el **PARSER** el cual clasifica y asigna función a cada expresión.

3. Interpretación del código una vez realizado el análisis por completo y mostrar en la interfaz gráfica el resultado.

Métodos Clase LEXER

- GetTokenList: crea lista de tokens
- GetToken: obtiene los tokens
- CompleteNumberLiteral: si el token empieza por un número, lo completa
- CompleteStringLiteral: si el token empieza con una "letra", lo completa
- CompleteWhiteSpace: al encontrar un espacio en blanco, mira su alcance

- CompleteKeyWordOrIdentifier: encuentra palabras claves
- CompleteSymbol: encuentra símbolos

Métodos Clase PARSER

- ParseCode: crea el árbol
- ParseStatementList: crea lista con los tipos de tokens
- ParseStatement: analiza cada token por separado.
- ParseLetInStatement: analiza la sintaxis let-in.
- ParseFunction: parsea declaración o llamado de función.
- ConvertToVariables: crea una lista con los nombres de las variables.
- ParseAssigment: analiza y devolver una asignación en función del token.
- ParseIdList: analiza una secuencia de tokens en busca de identificadores y almacenarlos en una lista.
- ParseFigureStatement: crea una instancia en función de la figura.
- ParsePredefinedFunctionCall: analiza una llamada a una función predefinida.
- ParseLogicalOr & ParseLogicalAnd: construye un árbol a partir de expresiones separadas por un "or" o "and" respectivamente
- ParseEqualityOperator: analiza expresiones separadas por "==" o "!=" .
- ParseComparison: realiza la comparación de expresiones matemáticas.
- IsComparerOperator: verifica que el token sea de tipo "Comparador"
- ParseSumOrRest: analiza expresiones de suma o resta.
- ParseMultiplicationorDivision: analiza expresiones de multiplicación o división.
- ParseNegativeOperator: analiza operador de negación en una expresión matemática.
- ParseNotOperator: analiza y devuelve una expresión que utiliza el operador de negación "not".
- ParseSequence: chequea el tipo de secuencia
- ParseRegularSequence: crea una secuencia de expresiones a partir de una lista de expresiones.
- ParseSequenceInRange: analiza una secuencia de números en un rango específico.

TIPOS DE ERRORES

Durante la compilación del programa pueden ocurrir los siguientes errores:

- **Error Léxico:** cuando la entrada presenta caracteres no válidos del lenguaje.
- **Error Sintáctico:** cuando la secuencia de tokens no es válida según la gramática del lenguaje.
- **Error Semántico:** cuando no se puede evaluar la expresión debido a incoherencias entre la operación y los tipos de expresiones que se encuentran en ella.

INTERFAZ GRÁFICA

A continuación se describe cada elemento de la interfaz gráfica:

- Botón **Compile & Run:** al dar clic en este botón se realiza la compilación y se visualiza ….
- Botón **Exit:** permite abandonar la ejecución del programa.

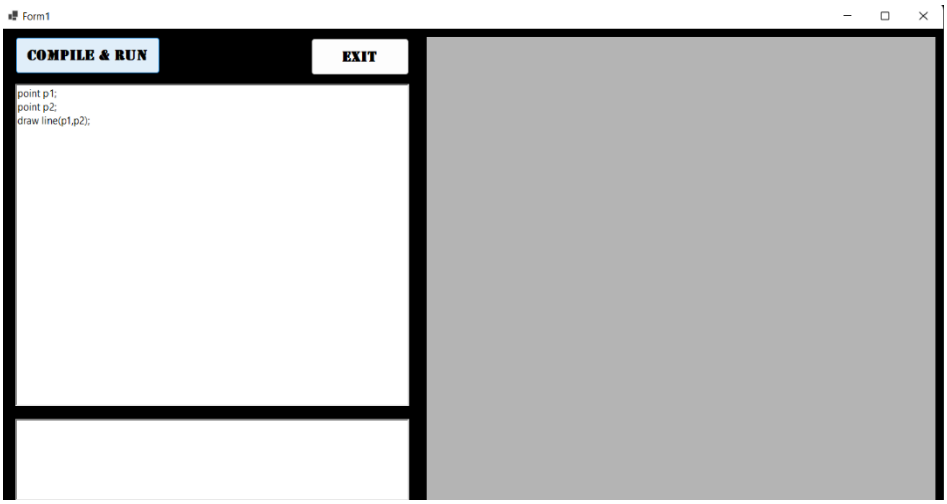
- TextBox (*rtb_Code*): área de la interfaz donde el usuario escribe el código.
- Área de graficado (*pctBox_Draw*)**:** área donde se grafica el resultado de las figuras correspondientes al código tecleado por el usuario.
- Área donde se visualizan los Errores (*rtb_Error*)**:** de producirse algún error en el proceso de compilación, es en esta área donde se visualiza y describe el texto del error producido.

COMANDOS Y FUNCIONES

Comando	**Descripción**
`point <id>`	Declara que se recibe un argumento de tipo punto con nombre <id>
`line <id>`	Declara que se recibe un argumento de tipo recta con nombre <id>
`segment <id>`	Declara que se recibe un argumento de tipo segmento con nombre <id>
`ray <id>`	Declara que se recibe un argumento de tipo semirecta con nombre <id>
`circle <id>`	Declara que se recibe un argumento de tipo circunferencia con nombre <id>
`point sequence <id>`	Declara que se recibe un argumento de tipo secuencia de puntos con nombre <id>
`line sequence <id>`	…
…	
`color`	Establece el color a ser utilizado
`restore`	Restablece el color anterior
`import <string>`	Incluye en el programa actual las definiciones del fichero indicado.
`draw <exp> <string>`	Dibuja el o los objetos definidos en <exp>
`line(p1,p2)`	Devuelve una recta que pasa por los puntos p1 y p2.
`segment(p1,p2)`	Devuelve un segmento con extremos en los puntos p1 y p2.
`ray(p1,p2)`	Devuelve una semirecta que comienza en p1 y pasa por p2.
`arc(p1,p2,p3,m)`	Devuelve un arco que tiene centro en p1, se extiende desde una semirecta que pasa por p2 hasta una semirecta que pasa por p3 y tiene medida m.
`circle(p,m)`	Devuelve una circunferencia con centro en p y medida m.
`measure(p1,p2)`	Devuelve una medida entre los puntos p1 y p2.
`intersect(f1,f2)`	Intersecta dos figuras (puntos, rectas, etc.) y devuelve la secuencia de puntos de intersección. Si la intersección coincide en infinitos puntos devuelve `undefined`.
`count(s)`	Devuelve la cantidad de elementos de una secuencia. Si la secuencia es infinita devuelve `undefined`.
`randoms()`	Devuelve una secuencia de valores aleatorios numéricos entre 0 y 1.
`points(f)`	Devuelve una secuencia de puntos aleatorios en una figura.
`samples()`	Devuelve una secuencia de puntos aleatorios en el lienzo.

EJEMPLO DE EJECUCIÓN DEL PROGRAMA

1- El usuario teclea el código



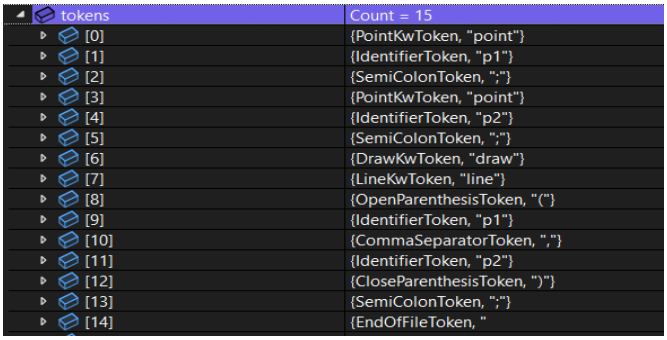
2- Una vez tecleado el código se procede a chequear a través de la clase **CallLogic** si hay que cargar alguna función predeterminada empleando a su vez, la clase **PredefinedFunction** (encargada de definir funciones predeterminadas) y el método **LoadSystemFunctions** perteneciente a la clase **CompilatorTools**.

3- Se procede a extraer la lista de Tokens mediante la clase **Lexer** donde se emplea el método **GetTokenList** el cual se apoya de los siguientes métodos:

- **GetToken**: para identificar el comienzo del token

- A continuación se hace un llamado a los métodos:

  CompleteNumberLiteral: el primer caracter es un número
  CompleteStringLiteral: el primer caracter es una declaración de string, es decir, comilla
  CompleteWhiteSpace: el primer caracter es un espacio en blanco, el cual no se guarda en la lista de string
  CompleteKeyWordOrIdentifier: el primer caracter es una letra
  CompleteSymbol: el primer caracter es algún símbolo



4- Al obtener la lista con los tokens, se comienza el parseo:

- Se llama al método **PrseStatementList** para realizar una lista con las declaraciones de cada token. Este método se apoya del método **ParseStatement** que según el tipo del token realiza su respectiva función.
- Mediante la clase **Assigment** se asignan a los tokens su respectivo tipo de expresión.
- Se recorre cada elemento del árbol y se obtiene su alcance.
- Se vuelve a recorrer cada elemento del árbol y se chequea su semántica, donde si existe alguna función como en este caso "draw", obtenemos sus variables y se verifica el tipo de figura se trazará.
- Una vez realizado todo este procedimiento se devuelve el resultado del árbol.

5- Al terminar de realizar el Parser, se recorre cada elemento del árbol y según su tipo de expresión se dirige a su respectiva clase para evaluarla.

6- Una vez evaluada, se muestra el graficado en el área de graficado