Facultad de Matemática y Computación

TERCER PROYECTO DE PROGRAMACIÓN

Colaboladores:

Joel Aparicio Tamayo
Claudia Hernández Pérez

Grupo: C-113

Lenguaje G-Sharp

Estructura:

La implementación del intérprete del lenguaje G# se ha realizado en un proyecto de tipo *Class Library* y consta de un conjunto de archivos en lenguaje C# que agrupan toda la lógica. Posee un *Lexer*, un *Parser*, un *Evaluator*, un *Checker*, y por supuesto toda la sintaxis de cada expressión del lenguaje. Las expresiones soportadas en el lenguaje son:

- **UnaryExpressions**: Expresiones unarias con los operadores +, -, not.
- **BinaryExpressions**: Expresiones binarias con los operadores $+, -, *, /, \%, and, or, <, <=, >, >=, ==, !=$.
- **LiteralExpressions**: Expresiones literales: números, strings, ...
- **ConstantExpressions**: Asignaciones de constantes y evaluación.
- **FunctionExpressions**: Asignaciones de funciones y evaluación
- **SequenceExpressions**: Secuencias finitas e infinitas
- **EncapsulatingExpressions**: Expresiones que encapsulan otras: paréntesis, let-in, condicionales.
- **StatementExpressions**: Expresiones void, import.
- **GeometryExpressions**: Expresiones geométricas: points, lines, ...

Lexing

El intérprete realiza el proceso de una forma bastante común: se recorre el texto caracter por caracter y se agrupan en *tokens*, que es el tipo básico para el análisis sintáctico. Si existe algún error por caracter inválido, se detecta y de devuelve un *ErrorToken*.

Parsing

El proceso de parsing convierte cada grupo de tokens en una expresión y devuelve un árbol sintáctico que contiene las expresiones a ser evaluadas.

El funcionamiento del *Parser* se basa en determinar según los tokens, qué tipo de expresión se está construyendo, obtener de ellos la información necesaria y devolver una expresión de tipo *IExpressionSyntax*. En caso de error, se devuelve un *ErrorExpressionSyntax*.

IExpressionSyntax

IExpressionSyntax es una interface que caracteriza a cada expresión del lenguaje. Posee los métodos de *Evaluate* y *Check* que garantizan que cada expresión pueda checkearse y evaluarse. Además contiene dos propiedades de solo lectura *Kind* y *ReturnType*, que permiten que cada expresión tenga un tipo general y un tipo según su evaluación, respectivamente.

Checker

El *Checker* realiza el chequeo semántico del árbol sintáctico que se generó en el proceso de parsing. Para ello simplemente se invoca al método *Check* de la expresión en cuestión y se retorna true o false.

Evaluator

Similar al *Checker*, el *Evaluator* solo invoca al método *Evaluate* de la expresión en cuestión, y se devuelve un object que es el resultado de dicha evaluación. De esto puede deducise que resultaría muy cómodo evaluar cualquier expresión nueva que se inserte al lenguaje, pues solo tiene que implementar *IExpressionSyntax*.

Scope

El otro objeto importante del programa (dejando a un lado las expresiones en sí) es el *Scope*, pues es donde se guardan los datos del entorno que engloba a la expresión en cuestión. El *Scope* almacena las constantes y funciones declaradas y sus respectivos valores o cuerpos.

Este objecto brinda muchas facilidades para anidar entornos hijos dentro de otros más globales. Por ejemplo, en el caso de los let-in, las instrucciones entre en *let* y el *in* se ejecutan con un *Scope* "hijo" que contiene los datos del *Scope* global más los nuevos datos que se guarden como resultado de las evaluaciones de dichas instrucciones. Esos datos desaparecen cuando se termina de evaluar el let-in, pues el *Scope* "padre" nunca fue modificado.

Interfaz gráfica (UI)

La interfaz gráfica está implementada en un proyecto de Windows Forms. Consta de dos formularios que interactuan constantemente uno con el otro.

Menú Principal

La primera interacción con el usuario la tiene "OpenWindow":
≫ "Start Drawing" ofrece acceso a la otra ventana,
≫ "More About" se muestra el informe del proyecto,
≫ "Exit" para salir de la aplicación.

Graficador

Esta ventana permite al usuario escribir en el lenguaje de G-Sharp en un textBox de donde se recibe la información para que el compilador trabaje. Para ejecutar el código debe compilarse y estar correcto antes de correrlo, de no ser así se imposibilita el acceso al botón "Run".
≫ "Compile" para compilar el código,
≫ "Run" para correr el código compilado y sin errores para dibujar,
≫ La sección de "Move it" permite desplazarte por el lienzo,
≫ La sección de "Zoom" permite aumentar y disminuir el tamaño de los elementos dibujados,
≫ "Reset Coordinates" resetea los cambios de desplazamiento y de escala,
≫ "Stop Drawing" si el usuario determina que quiere dejar de pintar un objeto que potencialmente es infinito este botón se lo permite,
≫ "View Files" para ver los documentos que están el carpeta files por defecto y los que se vayan creando,
≫ "Clear" reinicia el textBox y el pictureBox,
≫ "Save" guarda el código en un archivo txt para luego poder ser importado.