

**Instituto Tecnológico de Costa Rica**

**Campus San Carlos**

**Ingeniería en computación**

**Compiladores e Intérpretes – IC4700**

**Profesor: Oscar Víquez Acuña**

**Estudiantes:**

**Mariana Artavia Venegas**

**Francisco Javier Ovares Rojas**

**I Semestre 2023**

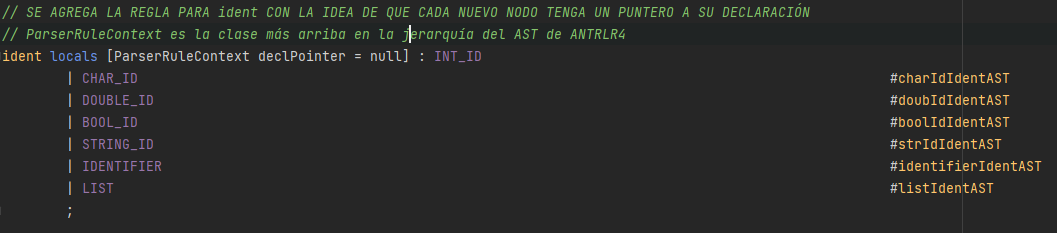
**Solución e Implementación**

Para la solución del presente proyecto se utilizaron dos herramientas clave:

* **ANTLR4**: Esta herramienta además de ser la base del proyecto incluso desde etapas anteriores, permite realizar modificaciones al Parser para facilitar manejo de los datos de las variables, los métodos y las clases posterior a su declaración implementando punteros a la dirección de memoria donde ocurrió la misma, asimismo mediante una propiedad booleana se indica si el elemento declarado es global o local. Por otro lado, ALTLR4 también permite generar fácilmente los visitors de cada una de las reglas declaradas en el Parser.
* **Reflection Namespace**: Esta librería de C# es la que permite la generación del bytecode o código ensamblador de .net.

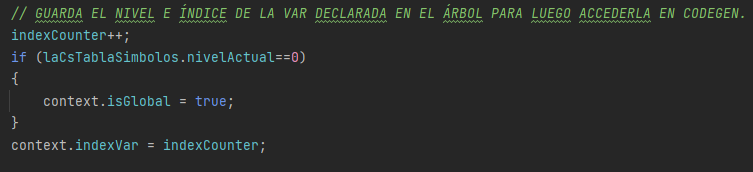
En primer lugar, como fue mencionado anteriormente se realizaron las siguientes modificaciones pertinentes en el archivo MiniCSharpParser.g4 mismo que contiene nuestro parser, siguiendo las recomendaciones y explicaciones dadas por el profesor.





Estas permiten tener un índice que funcione como un puntero a la dirección de memoria donde se almacena la variable declarada y el campo isGlobal permite identificar si la variable es global o local.

En segundo lugar, siguiendo la idea de implementación del índice se modifica la clase CSharpContextual, para que conforme realice la declaración de las variables se almacene su índice y se identifique si son globales o locales, añadiendo el siguiente código en VisitVarDeclaAST:



Tal y como se indica en el comentario, esta información será accedida en la clase CodeGen misma que constituye el siguiente paso de la solución. Esta clase es la encargada de la generación del bytecode, misma acción que se realiza por medio de sobrescribir los visitors de las reglas del parser. Después de inicializar esta clase con todos los elementos necesarios para implementar el uso de la Reflection, se procede a la codificación de los visitors basándose en las siguientes plantillas de código.

program : (using)\* CLASS ident LEFTBRACK (varDecl | classDecl | methodDecl)\* RIGHTBRACK EOF #programAST;

En el caso de la regla regla program se toma en cuenta que la implementación de reflection conlleva la creación de una “clase” un type que alberga todo el código generado por lo que luego de visitar todos los elementos del program se crea una instancia de esta clase y se establece como punto de entrada el método main.



En el caso de las variables, primero se inicializan tomando en cuenta el “tipo”, con esto se crea una variable ya se local se usan LocalBuilder en el cual se especifica el tipo con el tipo para posteriormente asignarle a esa variable valores del mismo. Si es global se utiliza FieldBuilder y se le especifican más cosas como los atributos (public, private y statatic) el tipo y el nombre de la misma. En el caso de las locales se manejó el nombre mediante un diccionario para luego accederlo. Se carga la variable en la pila y luego se asigna el valor a la misma, el cual luego puede ser consultado, como en el ejemplo de imprimirl el valor de una variable mediante el método write.

|  |  |
| --- | --- |
| **Plantilla** | |
| **Código** | **Texto fuente Ejemplo** |
| Clases | class Ejemplo {} |
| Declaración de Variables | int varEjemplo, anotherVar;  string varDos;  bool isGlobal;  char firstLetter; |
| Asignación de Variables | varEjempo = 10;  varDos = “Hola Profe”;  ... |
| Declaración de Métodos | void main(){} |
| Código de métodos | void main()  {  int varEjemplo;  varEjemplo = 10;  } |
| Llamado a Métodos | method();  Ejemplo:  write(varDos); |

**Análisis de resultados**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Actividad o tarea** | **Codificación** | **Funcionalidad** | **Observaciones** | |
| 1. Implementación de la librería Reflection. | 100% | 100% |  |
| 1. Declaración de clases y sus respectivos atributos. | 20% | 20% | No se implementó la declaración de clases, solamente la declaración de atributos globales es decir a nivel de la clase principal. |
| 1. Declaración de métodos. | 100% | 100% |  |
| 1. Declaración de variables. | 100% | 80% | Nose verificó la declaración de variables de tipos complejos. |
| 1. Asignación de variables. | 100% | 100% |  |
| 1. Llamada a métodos. | 50% | 50% | Se implementó la llamada a métodos cuando estos son usados en asignaciones o enviados a un write. No las llamadas generales, tampoco se realizó en manejo de parámetros, por lo que solo funcionan los void. |
| 1. Codificación de if, while, for, entre otras estructuras. | 100% | 90% | Esta la funcionalidad de estos no fue probada por falta de otros elementos. |
| 1. Implementación del método write. | 100% | 90% | Podrían presentarse problemas relacionados con la impresión de métodos. |
| 1. Implementación del método pre declarado len para arreglos | 0% | 0% | No se implementó. |
| 1. Implementación del método read. | 0% | 0% | No se implementó. |
| 1. Codificación de los operadores, relop, addop, mulop. | 100% | 80% | No se probó su funcionamiento. |

**Conclusiones**

1. Se logró tener un acercamiento con lo que viene siendo la generación de bytecode, la experiencia fue un poco compleja al no tener pleno conocimiento del “lenguaje” es decir los emits y demás instrucciones, sin embargo, fue una experiencia valiosa ya que nos permitió tener una visión más clara del propósito general del proyecto y del concepto de compilador.
2. Al desconocer el funcionamiento y la correcta implementación del lenguaje reflection fue complicado reconocer los errores que se generaron durante la codificación del proyecto, por lo que se presentaron grandes atrasos en el desarrollo del proyecto.
3. Se realizó una implementación exitosa de la generación de código de varias de las funcionalidades básicas de nuestro lenguaje Mini C#, estos resultados positivos a pesar de que el producto como tal no es completamente funcional como se esperaba, fueron gratificantes ya que permitieron ver materializado el trabajo y esfuerzo invertido en el compilador como producto a lo largo del semestre.

**Bibliografía**

Dotnet-Bot. (n.d.). *System.Reflection Namespace*. Microsoft Learn. <https://learn.microsoft.com/en-us/dotnet/api/system.reflection?view=netframework-4.8>