*Instituto Tecnológico de Costa Rica*

*Sede Regional San Carlos*

*Escuela Ingeniería en Computación*

*Compiladores e Intérpretes*

# ***Tarea Programada #2***

# ***"Mini - C# -- Contextual"***

*Responsable:*

*Hellen Rojas Rojas 2013083934*
*Henry Solís Chacón 2013085706*

*4 de mayo del 2016*
*Santa Clara, San Carlos*

## Soluciones e implementación

Para la implementación del proyecto se trabajó en análisis contextual de código, para esto inicialmente se crearon tres tablas, la primera llamada tabla de símbolos, esta guarda las variables, constantes y parámetros, a la misma se le hicieron métodos de inserción, búsqueda y uno de eliminación que limpia la tabla una vez que se cierra un nivel, luego se tiene la tabla de métodos que contiene las funciones y el tipo de los parámetros de las mismas, además contiene métodos para insertar y buscar funciones, después tenemos la tabla de clases internas que contiene las clases y sus variables, a esta tabla se le hicieron métodos de inserción de clases y variables, y métodos de búsqueda.

En la implementación de las tablas se hicieron los visitor, primero se tiene el de program que en general visita las variables globales, constantes, métodos y luego en el cuerpo de la clase general se visitan los métodos. El visitor de class verifica que no exista una clase definida previamente con el mismo Id y que sus variables no tengan el mismo nombre, y además inserta las clases y sus variables para luego accederlas. Seguidamente se tiene el visitor de declaración de variables que lo que hace es verificar que estas variables no hayan sido declaradas anteriormente y el nivel y luego las inserta en la tabla, además se tiene el visitor de constantes que es muy similar solo que se hacen las asignaciones al mismo tiempo y se verifica que los tipos concuerden. En cuanto al visitor del método tiene un poco más de complejidad, primero se verifica por medio de la tabla que no exista otro método con el mismo nombre, una vez que se verifica esto, se recorren los parámetros (verificando tipos y demás) para incluirlos en la tabla de símbolos, una vez que se ha hecho esto se recorre un bloque que contiene una serie de declaraciones distintas, a continuación se describe de manera muy general algunas de ellas.

**Visitor de designación (`designatorStatAST`):** En esta se puede hacer asignaciones, llamada a métodos, incremento o decremento de variables, hacer *new* de arreglos y clases y acceder a atributos de las clases en general. En las asignaciones se verifican que los tipos sean iguales, en las llamadas a métodos se comprueba que tenga el número y tipo de argumentos correctos consultándolo en la tabla de

métodos y asimismo que el método exista en la tabla, en cuando al incremento o decremento se verifica que sean de tipo número. Además verifica que no se pueda hacer *new* a clases o arreglos inexistentes o acceder a variables de clases inexistentes.

**Visitor de condición (`ifStatAST`):** Primero se verifica que la condición este correctamente, algunos aspectos a verificar en la condición son que los tipos en la comparación sean iguales, que no se puedan comparar con < , >, >=, <= u otra cosa que no sean números, luego de esto se puede tener código dentro, el cual puede ser más designaciones, además también se contempla la posibilidad de tener *else*.

En cuando a los visitor de los ciclos son muy similares al visitor de condición, pero en el caso del foreach se verifica que la variable sea del mismo tipo del arreglo y que el primero argumento sea una variable y el segundo un arreglo.

Luego de estos se tienen algunas otras posibles declaraciones como break en el cual no se verifica nada, el return en el cual se verifica que sea del mismo tipo de la función, Write, Read… etc.

Además cabe resaltar que luego del visit del bloque se verifica que el método tenga retorno si el tipo no es void.

| Objetivo | Estado | Descripción |
|---|---|---|
| **Tabla Símbolos** | Completo | |
| **Tabla de Métodos** | Completo | |
| **Tabla de Clases Internas** | Completo | |
| **Métodos Preestablecidos** | Completo | |
| **Operadores Aritméticos** | Completo | |
| **Operadores Relaciones Binarias** | Completo | |
| **Arreglos Y Clases** | Completo | |
| **Identificador no declarado 2 veces** | Completo | |
| **Identificador no usado 2 veces** | Completo | |
| **Chequeo tipos Expresiones y Statements** | Completo | |

## Resultados Obtenidos

El lenguaje utilizado en el proyecto es una pequeña parte de C#, en el mismo podemos observar diferencias como la declaración de variables fuera del bloque que afecta durante la revisión contextual, dado que al almacenarlos para su futuro uso o revisión complican la búsqueda y conservación del nivel.

También hay diferencias como el uso de arreglo de dos dimensiones, debido a que el lenguaje inicial planteado indicaba solo el uso de un arreglo simple, limitando el lenguaje respecto a su original.

Los errores se intentan mantener acorde a los originales de C#, pero hay que considerar que no se implementaron todos dado que hay limitaciones en el lenguaje, por otro lado C# cuando encuentra un error no se detiene y los muestra todos al final, siguiendo esta lógica nuestro Compilador realiza lo mismo.

## Conclusiones

El proyecto es una buena forma de comprender a fondo el trabajo de un compilador, en este caso desde la perspectiva contextual, el proyecto es largo y consume mucho tiempo dado que hay que pensar las formas de almacenar las variables, métodos, clases, entre otros, conlleva a un análisis importante dado que de ello depende totalmente el avance del proyecto. Además hay que tomar en cuenta la necesidad de tener conocimiento sobre C# o los errores normales (en contextual).

La curva de aprendizaje no es tan alta dado que por la ubicación del curso en la malla curricular, ya se debe tener conocimientos base para trabajar este tipo de lenguajes, aunque el uso del ANTLR4 si representaba algo de aprendizaje pero no significativo.

En general el proyecto brinda una gran cantidad de aprendizaje y puesta en práctica de los conocimientos adquiridos en el mismo.

## Bibliografía

ANTRL4. (04 de 05 de 2016). *Antrl4*. Obtenido de http://www.antlr.org/

Microsoft. (04 de 05 de 2016). *Microsoft*. Obtenido de https://msdn.microsoft.com/es-es/library/ms173163.aspx