Informe de Trabajo Práctico Nº 2

22 de abril de 2016

75.42 Taller de Programación I

Cátedra Veiga

Facultad de Ingeniera - UBA

Autor: Martín Stancanelli

Padrón: 95188

Objetivo

El objetivo de este trabajo practico fue la implementación de un compilador de LISP con multithreading, utilizando la librería pthread, y aplicando conocimientos sobre programación orientada a objetos.

Configuración del entorno

El entorno de trabajo elegido para la realización del trabajo practico fue una PC con SO Linux Mint 17.3 y compilador gcc 4.8 configurado para utilizar el estándar de C++98 y la librería pthread.

Explicación de enunciado

El trabajo consiste en realizar un compilador de LISP que permita enviar instrucciones de LISP por la entrada estándar, hacer el correspondiente análisis sintáctico y semántico, y ejecutarlas. Como novedad se incorpora la capacidad de realizar el proceso de compilación en paralelo, osea con la utilización de threads.
El programa parseara y compilara linea a linea. En caso de encontrar una expresión *Sync* esperara a los threads que estén ejecutándose en ese momento para continuar. Se repite el proceso hasta que se encuentra el fin de archivo.

Desarrollo

La parte mas interesante en términos de diseño del trabajo practico es la estructura de clases de las expresiones y funciones de LISP. En un principio se encaro el trabajo con un enfoque bottom-up, generando las clases de funciones de mas bajo nivel (*Sum, Divide*, etc.). Luego, al ver la complejidad de la situación y la necesidad de implementar el parser recursivo de la solución que finalmente se adopto, fue necesario tomar un enfoque mas global, es decir, empezar por planificar el manejo de *Expressions* como algo general y luego una vez que esto funcionase si especificar el modelo hacia abajo.

Entonces, se creo la clase *Expression* y se modeló todo el parser con esta clase. Luego se creo la clase *Constant* y una función simple como *Sum*, que permitieran probar parseos de lineas simples. Una vez resuelto este problema y al notar que había funciones que compartían varias características con la función *Sum* se creo la clase *Function*. Como existe la necesidad de manejar todo en forma general se hizo heredar a las clases *Function* y *Constant* de la clase virtual *Expression*. Esto ultimo permite que el parser solo maneje *Expressions* y luego cada una de ellas a través del polimorfismo sepa como evaluarse.

Se incluye un diagrama de clases (1) explicando la estructura final del modelo.

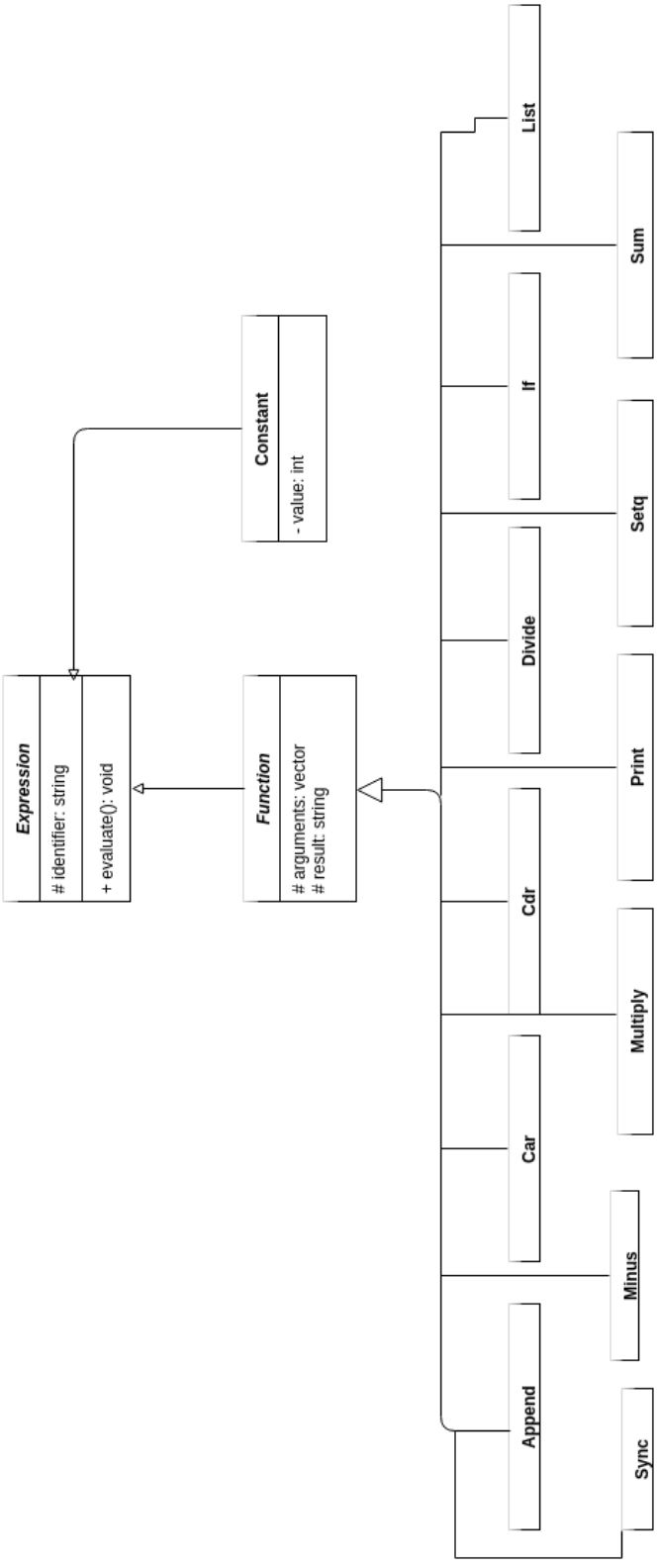

Figura 1: Diagrama de clases

Conclusión

Este trabajo necesitó de la utilización de conocimientos bastante variados y complejos. Que la solución final propuesta utilice recursividad le agrega una complejidad considerable, por el hecho de que esta hace mas difícil el seguimiento de la elaboración del código, por ejemplo al querer debuggear las estructuras recursivas que se generan. No obstante, una vez resuelto el problema recursivo, el modelado de las funciones salio de forma natural y la estructura que se genero permitía que agregar nuevas funciones el programa no llevara mas de unos pocos minutos.
Finalmente se implemento la parte de multithreading. Este era un conocimiento nuevo, el cual tuvo que ser debatido con otros compañeros del curso para lograr entenderlo del todo. Pero una vez que la idea general estaba pulida no presento extrema complejidad.