Compiladores

Martínez Coronel Brayan Yosafat

Funciones y procedimientos.

3CM7

4/1/2021

Práctica 7: Robot

Robot con funciones y procedimientos

Introducción

El objetivo de esta práctica es agregar funciones y procedimientos a lo que se ha construido a lo largo de las otras seis prácticas. Esta sería la culminación de funcionalidad, excepto, que si fuera un lenguaje de programación que se sacara al público entonces lo de arriba sería algo completamente falso, ahondaremos en ello en la conclusión, por ahora, presentemos los puntos específicos a los que se tuvieron que completar para decir que está terminada la práctica:

- Nuevos tokens sobre FUNCTION FUNC ARG RETURN PROC PROCEDURE
- Función para guardar la función
- Función para llamar una función
- Hacer que las producciones permitan recursión
- Hacer que los argumentos se guarden bien en el frame
- Nueva clase Frame
- Nueva pila de Frames
- Funciones para el flujo de la llamada de función
- Producciones de los argumentos
- Forma de llamar al enésimo argumento
- Unificar las producciones para aumentar la utilidad de funciones y procedimientos

Por supuesto, hubo en realidad muchos más puntos específicos, pero no lo suficientemente grandes como para ser nombrados en la lista anterior. Lo que cabe resaltar es que agregamos una nueva clase. En las prácticas anteriores solamente utilizábamos casi las mismas clases para todos lo que requeríamos, con excepción, por supuesto, de la práctica en donde agregamos la máquina y la tabla. Por supuesto, también se agregó la pila de llamadas, que permite recursión junto con las producciones en el archivo Y. Ahora bien, en el desarrollo explicaremos cómo funciona el mecanismo de llamada de una función y cómo es que se guardan las funciones dentro de la máquina.

Desarrollo

Primero comencemos con la clase Frame, contiene 3 cosas importantes, el índice de donde debe regresar en cuanto termine la llamada de una función, el número de argumentos que contiene y los argumentos, que siempre serán secuencias, en otras palabras, son cadenas de caracteres. Esto es sumamente importante porque un argpush no es realmente distinto de un varpush porque sólo cambia su origen. Sin embargo, con fines de claridad se separan las funciones para que a la hora de ver la pila se vea muy claro a dónde pertenece y de dónde salió esa secuencia.

Ahora, podemos hablar sobre cómo es que se guarda una función, para eso tenemos un vector de funciones que puede contener de la misma forma que donde se guarda el código generado por el flujo común, pero este será guardado para que no se pierda en cada una de las ejecuciones. Ahora, en la tabla de símbolos se pueden guardar… símbolos, que pueden tener un valor, en este caso es usado como nombre y que pueden tener un entero, que en las funciones y procedimientos son utilizados como índice de dónde debe de empezar la función. Como se había mostrado antes se utiliza STOPFUN para marcar en dónde acaba cierto pedazo de una función o de una estructurad de control.

De hecho, en la función de guardar una función se usa un entero para saber cuántos STOPFUN debe de leer para determinar que una función ya está completamente en las funciones, los valores son los siguientes:

- Primero el contador se inicia en 1, porque debe de ver al menos uno
- Si ve un if, entonces debe agregar 3 al contador, ya que se usa para el bloque, la lógica y el bloque del else.
- Si ve un queue, entonces es que el else no existe en un if, y le resta 1
- Si ve un while, entonces obligatoriamente tendrá que ver dos STOPFUN más
- Si ve un for, entonces verá dos STOPFUN, el del rango y el del bloque

Ahora, como cuando ve una cadena de caracteres no reconocida en la tabla de símbolos, entonces siempre crea una variable, pero cuando haga eso ahora puede ser una función o un procedimiento. Entonces, las producciones se hicieron de forma que al inicio pueda poner una

variable en el nombre del procedimiento o función. Luego, si se desea reescribir la función o procedimiento, entonces la producción también cubre eso. En ambos casos se llama a savefun para que guarde la función.

Ahora bien, las producciones se hicieron para soportar la recursividad, un ejemplo de eso es el siguiente código que se muestra en el video de demostración:

```
dibujar on ;

proc rec ( ) {

        S S ;

        rec ( ) ;

}

rec ( ) ;

proc rec ( ) {

        N ;

        rec ( ) ;

}
```

Mientras que un ejemplo de una función también es parte de las expresiones porque obligatoriamente debe de tener un return al final de su escritura, es el siguiente:

```
fun suma ( ) {

        return $:1 + $:2 ;

}

fun doble ( ) {

        return suma ( S S , E E ) ;

}
```

```
var = doble ( ) ;

var ;
```

Conclusiones

Como se veía venir, aumentamos mucho la funcionalidad de nuestro robot, sin embargo, como lo dijimos arriba, si este fuera un lenguaje de programación que sacáramos al mundo, entonces el aumento que hemos dado solo sería un comienzo, podemos dar sintaxis a muchas más cosas, podemos hacer muchas más características especiales, podríamos hacer compatibilidad con muchos otros lenguajes, podríamos crear cosas especiales como un conector a una base de datos, como los que existen en muchos lenguajes, crear con el apoyo de la comunidad un nuevo lenguaje. Ha sido muy interesante ver que el open source es muy importante, y que es sumamente importante para la propia comunidad. El talento está escondido tras cada hogar, el talento es anónimo muchas veces, alguien dedicó su tiempo en hacer lenguajes como los tenemos ahora, quizá no me hubiera parecido tan importante antes. Sé que lo que hice es algo sumamente pequeño comparado con lenguajes que tienen años en la comunidad, pero, estoy seguro que los avances pequeños de personas como yo, son lo que hicieron que Python esté tan arriba en la lista de lenguajes usados, cuando un lenguaje no se hace abierto, cuando un lenguaje se pretende cobrar al usar, entonces solo queda su final, porque el talento colectivo es mucho más importante que el talento comprado.