72.39 - Autómatas, Teoría de Lenguajes y Compiladores

Comisión S

30 de junio del 2019

Trabajo Práctico Especial

FerLang

Autores

Jimena Lozano 58095

Emilio Basualdo 58172

Fermín Luciano Gómez 58111

Maite Herrán 57609

Índice

Idea subyacente y objetivo del lenguaje

La idea principal del equipo fue desarrollar un lenguaje cuyo objetivo es generar una mayor fluidez a la hora de escribir código, logrando que el programador no pierda tiempo pensando en la declaración de variables o funciones, y que otros programadores puedan leer y entender el código del otro rápidamente. Este lenguaje al mismo tiempo sostiene las condiciones básicas de un lenguaje para desarrollar programas, como bloques condicionales y ciclos.
Se tomó la decisión de que al momento de definir una variable, esta se pueda definir como una constante usando *'let' o* como una variable usando *'var'*, seguido de un nombre que la represente. Como se puede notar, no se ha mencionado nada sobre qué tipo de dato guardará, a diferencia de otros lenguajes como C, Java, C++, etc.
Esto no es indicio de que FerLang no es un lenguaje tipado. FerLang es un lenguaje tipado pero, a la hora de definir un tipo de variable, podemos optar por que el compilador infiera el tipo de la variable al asignarle un valor a la misma. De todas maneras, contamos con la opción de que el programador pueda definir un tipo de variable al crear una variable.
Ejemplo:

```
start
      var x = 10;
      var y = "hola mundo";
      var z: number;
      var n:number = 10;
      let j  = 5.89;
end
```

Se le dió mucha importancia a que las palabras reservadas para el leguaje sean significativas y correspondan con la operación que realizan y familiares, facilitando la adaptación a la hora de pasar de otro lenguaje a FerLang.
Por ejemplo, palabras como:

- `put`
- `getnum`
- `getchar`
- `getdec`
- `start`
- `end`

Consideraciones realizadas (no previstas en el enunciado)

Para lograr un lenguaje más completo, decidimos definir otros tipos de variables además de los requeridos por la cátedra, como decimal o caracter.
Por otra parte, para lograr un buen feedback al programador, desarrollamos una mejor definición de errores para indicarle con más detalle qué tipo de errores tiene el código.

Por ejemplo, se notan errores como redefinición de variables, variables inexistentes, y asignaciones incorrectas.

Descripción del desarrollo del TP

En base a nuestros objetivos para el lenguaje, tuvimos que tomar decisiones y diseñar nuestro propio lenguaje, comportamiento y gramática.
Una vez pasada la etapa de diseño recurrimos al material de consulta e internet para desarrollar un analizador léxico con Lex que identifique los lexemas de nuestro lenguaje, definidos por la gramática. Luego, con el uso de Yacc, se establecieron las reglas de la gramática. En esta etapa se encontraron errores en cuanto el diseño, por lo cual se tuvieron que redefinir varias ideas con respecto a la idea original que fue planeada.

Descripción de la gramática

Tipos

Tipo	Ejemplo

Números enteros	var x:number = 5;
	var x = 5;
	var x:number;

Números decimales	var x:decimal = 2.5;
	var x = 2.5;
	var x:decimal;

Cadena de caracteres	var x:string = “hola mundo”; var x = “hola mundo”; var x:string;
Carácter	var x:character = ‘a’; var x = ‘a’; var x:character;

No existe el tipo de variable booleana, pero contamos con la asignación de constantes true/false a variables, que serían traducidas a los enteros 1 (true) y 0 (false).

Delimitadores

Delimitador	Descripción	Ejemplo

start, end	Delimita el comienzo y final de un programa	start ...*código*... end
;	Delimita el final de una sentencia de código	var x:decimal; getdec(x);

{ }	Delimita un bloque de código	while (true) {
		...*código*....
		}

( )	Delimita una expresión booleana	if (5 <= 6)
,	Delimita una variable de la otra, utilizado en la función “put”	let text = "Enter a number"; put ("%s", text);

Comentarios

Los comentarios deben estar delimitados por ‘#’.
Por ejemplo:
esto es un comentario y acá puede ir cualquier cosa, tanto números 9 13 52.8 como símbolos : ‘ > * ^ ~

Operadores

Operador	Descripción	Ejemplo
+	Calcula la suma	var x = 1 + 5;
-	Calcula la resta	var x = 1 - 5;
*	Calcula la multiplicación	var x = 1 * 5;
/	Calcula la división	var x = 1 / 5;
%	Calcula el módulo	var x = 1 % 5;
++	Suma en 1 el valor de una variable	var x:decimal= 2.5; x++;
--	Resta en 1 el valor de una variable	var x:number = 1; x--;
<	Evalúa si la expresión a derecha es la mayor	(x < y)
>	Evalúa si la expresión a	(x > y)

	izquier es la mayor	
<=	Evalúa si la expresión a derecha es la mayor o igual	(x <= y)
>=	Evalúa si la expresión a izquier es la mayor o igual	(x >= y)
==	Evalúa si las expresiones a ambos lados del operando son iguales	(x == y)
!=	Evalúa si las expresiones a ambos lados del operando son distintas	(x != y)
or	Evalúa si alguna de las expresiones a ambos lados del operando es cierta	(x < y) or (x > y)
and	Evalúa si las expresiones a ambos lados del operando son verdaderas	(x < y) and (y < z)
not	Negación de una expresión	not (x < y)

Ciclos

Ciclo	Ejemplo

While	while (j <= i) {
	put ("* ");
	j++;
	}

Do - While	do {
	put ("* ");
	j++;
	} while (j <= i)

Condicionales

Condicional	Ejemplo
If	if (j <= i) {

	put ("* ");
	j++;
	}

If - Else	if (j <= i) {
	put ("* ");
	j++;
	} else {
	put ("^ ");
	j++;
	}

STDIN y STDOUT

Funcion	Descripción	Ejemplo
put	Escribe en la salida estándar acorde a un formato pasado por parámetro como una cadena de caracteres. Seguido de variables de donde obtener datos	put(“hola mundo”); var name = “gabriel”; put(“hola %s”, name); put(“el decimal es %lf”, 2.5);
getdec	Lee de la entrada estándar un decimal y lo guarda en la variable pasada por parámetro	var x:decimal; getdec(x);
getchar	Lee de la entrada estándar un carácter y lo guarda en la variable pasada por parámetro	var x:character; getchar(x);
getnum	Lee de la entrada estándar un entero y lo guarda en la variable pasada por parámetro	var x:number; getnum(x);

Benchmarking

Creamos un benchmark de primalidad, el cual mide el tiempo que le toma a un programa escrito en FerLang y a otro escrito en C en encontrar todos los número primos entre el 0 y el 500.000.
Las instrucciones para compilar y ejecutar el benchmark se puede encontrar en el README.md

El siguiente es el código en Ferlang

```
start
  var i = 1;
  var upper = 500000;
  var j : number;
  while (i < upper) {
      j = 2;
      while (j <= i/2) {
        if(i % j == 0) {
            stop;
         }
        j++;
      }
      i++;
  }
end
```

El siguiente es el código en C

```
int main(int argc, char const *argv[]) {
  int i = 1;
  int upper = 500000;
  int j;
  while (i < upper) {
      j = 2;
      while (j <= i/2) {
        if(i % j == 0) {
            break;
         }
        j++;
      }
      i++;
  }
  return 0;
}
```

Estos fueron los resultados(corridos en MacBook de serie C02K56Z4FFT0, MacOs 10.12.6, 2.4 GHz Intel Core i7, RAM 8 GB 1600 MHz DDR3) medidos con el comando *time.*

Lenguaje	Medición	Corrida 1	Corrida 2	Corrida 3
C	real	0m26.118s	0m27.648s	0m25.811s
	user	0m26.103s	0m27.443s	0m25.798s
	sys	0m0.014s	0m0.080s	0m0.012s
FerLang	real	0m25.788s	0m26.475s	0m25.770s
	user	0m25.780s	0m26.386s	0m25.758s
	sys	0m0.010s	0m0.035s	0m0.011s

Los resultados son muy similares pues el código en C generado por el compilador de FerLang es casi idéntico al código C que se utiliza para comparar.

El siguiente es el código en C, generado con el compilador de FerLang, a partir del código escrito en FerLang. (indentaciones a mano para la lectura)

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
  int i = 1;
  int upper = 500000;
  int j;
  while(i < upper) {
    j = 2;
    while(j <= i / 2) {
      if(i % j == 0) {
        break;
      }
      j++;
    }
    i++;
  }
}
```

Dificultades encontradas en el desarrollo del TP

Al compilar el proyecto, YACC en su salida muestra warnings. Uno de ellos es un error de reducción/desplazamiento el cual YACC intenta resolver por su cuenta.

Estos warnings eran generados por los operadores, para solucionar el problema se les asignó precedencia a los mismos en la gramática utilizando la opción “%left”. Sin embargo por alguna razón los warnings no cesaron, no conocemos el motivo de esto pero confiamos en que YACC va a saber resolver el error de procedencia correctamente.

Limitaciones y futuras extensiones

- **Arrays**

  Contamos con lectura de strings, un array de caracteres, pero no contamos con la declaración de variables de arrays para cualquier tipo.

- **Punteros**

  Un elemento de gran utilidad, que resulta indispensable para los programadores, es el puntero. Referirnos al valor de otra variable sería una gran extensión para el lenguaje.

- **Manejo de memoria**

  No contamos con funciones de manejo de memoria, como el *malloc*, *realloc*, *calloc*, *memcpy*, etc, del lenguaje C, por lo que el manejo de memoria es relativamente pobre. Para programar con mayor eficiencia en cuanto a memoria, esta extensión sería ideal.

- **Get strings**

  Leer strings de STDIN es una futura extensión para darle más opciones al programa en cuanto a la lectura de STDIN, ahorrandole al programador esfuerzos para leer input del usuario. En particular esta extensión se nos hizo necesaria a la hora de programar benchmarks dado que queríamos comparar tiempos de diversos escenarios de entrada.

Referencias

- https://m.youtube.com/watch?v=54bo1qaHAfk&list=PLkB3phqR3X43IRqPT0t1iBfmT5bvn198Z
- https://github.com/abhilash1in/rit-code/blob/master/yacc.y
- https://github.com/petewarden/c_hashmap