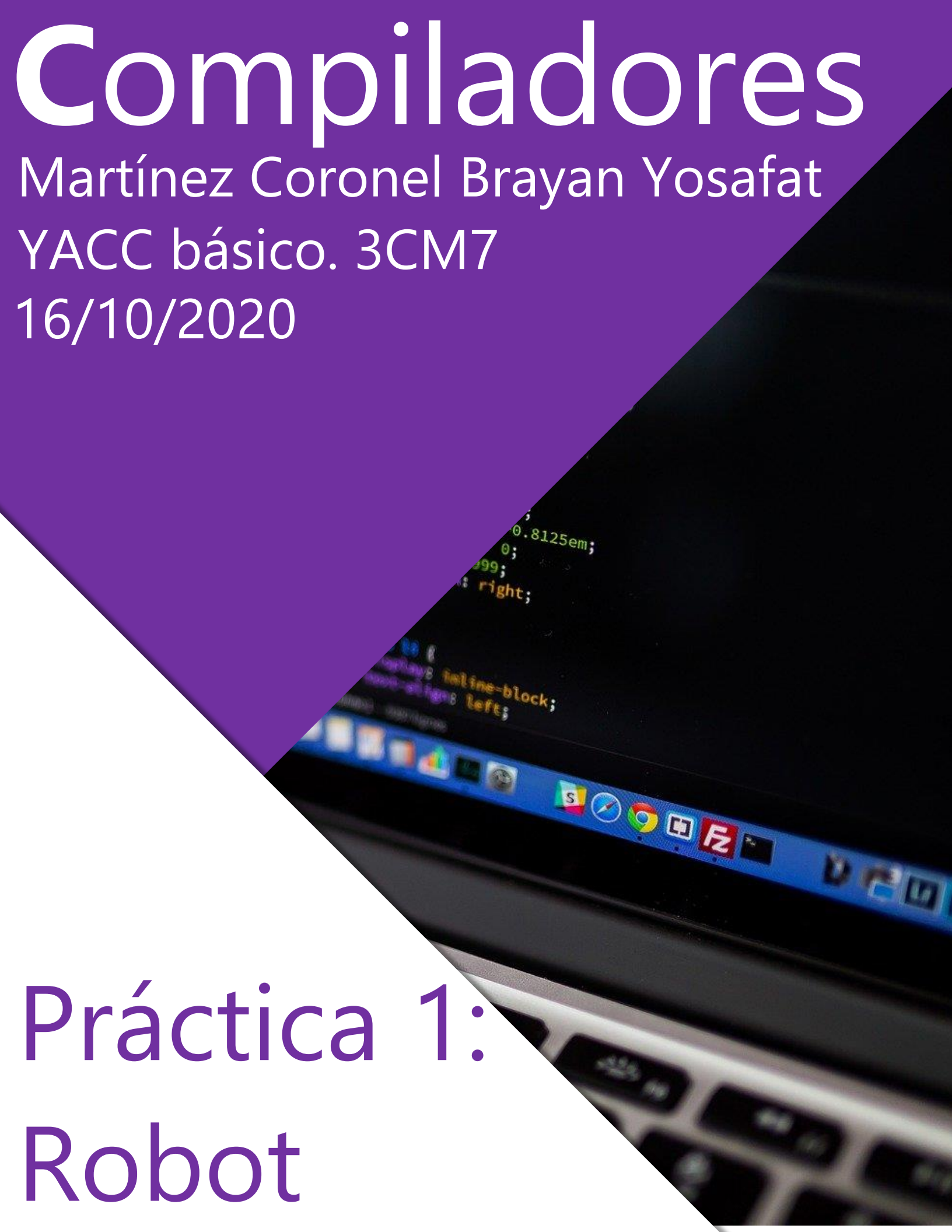
Compiladores

Martínez Coronel Brayan Yosafat

YACC básico. 3CM7

16/10/2020



Práctica 1: Robot

YACC básico: Robot

Introducción

Lo que debemos de tener en cuenta es qué necesitamos y qué queremos lograr, de esta forma será mucho más fácil diseñar una solución y que de verdad cumpla con lo que nos están solicitando. El problema enuncia lo siguiente: Escribir una especificación de YACC para calcular e imprimir la posición final de robot. También nos ofrece una tabla con las producciones y reglas semánticas que requeriremos:

PRODUCCIÓN	REGLA SEMÁNTICA
sec -> comienza	*sec.x := 0* *sec. y :=* 0
sec -> sec instr	*sec.x := sec.x + instr.dx* *sec. y := sec.y + instr.dy*
instr -> este	*instr.dx := 1* *instr.dy := 0*
instr -> norte	*instr.dx:= 0* *instr.dy:=1*
instr -> oeste	*instr.dx := -1* *instr.dy := 0*
Instr -> sur	*instr.dx := 0* *instr.dy := -1*

Pero, esto en realidad, aunque nos ayuda mucho, le falta algo, y es que instrucción termina, así que le agregamos una producción más: sec -> instr termina. (Para saber cuándo termina en vez de tener que imprimir todas las coordenadas del robot).

Desarrollo

Ahora, lo que tenemos al alcance, es el código llamado hoc1.y que, si bien es sumamente útil, tenemos que cambiar muchas cosas para que quede como lo deseamos, listemos las cosas que requerimos:

- Debemos usar una estructura de C, que tenga a x y a y
- El tipo de dato de la pila de yacc debe ser el de la estructura
- En vez de definirlo directamente, debemos usar %union para hacer la estructura
- En las acciones gramaticales deben manejar dos atributos, la manera de hacerlo es mediante una estructura de C

Ahora pasemos a la implementación, para hacerlo mucho más fácil, procederemos a explicar pedazo por pedazo del código, además, son menos de 100 líneas, así que será relativamente sencillo:

```
 8 %{
 9         #include <stdio.h>
10         #include <stdlib.h>
11         #include <math.h>
12
13         void yyerror (char *s);
14         int yylex ();
15         void warning(char *s, char *t);
16 %}
```

En estas líneas ponemos las declaraciones, lo importante es que YYSTYPE no está y que sí está <stdlib.h>, explicaremos por qué en el siguiente bloque de código.

Como en realidad, no se hacen operaciones con lo que entra, sino con 'comandos' predefinidos, entonces todo es un token. Lo importante es el uso de unión, de hecho, YYSTYPE sale automáticamente de ahí, aquí también está la estructura en C que necesitamos, como todos son de un tipo especial, usamos los menores que y los mayores que para poner qué tipo son (en este caso coords).

```
18 %union {
19         struct {int x, y;} sec;
20         struct {int dx, dy;} instr;
21 }
22
23 %token<sec> COMIENZA
24 %token<instr> ESTE
25 %token<instr> NORTE
26 %token<instr> OESTE
27 %token<instr> SUR
28 %token<sec> TERMINA
29
30 %type<sec> sec
31 %type<instr> instr
```

La gramática de hecho queda bastante similar a lo que está en la tabla:

```
34 %%
35 sec     : COMIENZA      { $$.x = 0; $$.y = 0; }
36         | sec instr     { $$.x = $1.x + $2.dx;
37                           $$.y = $1.y + $2.dy; }
38         | sec TERMINA   { printf("Nuevas coordenadas: %d,%d\n",
39                           $$.x, $$.y); exit(0);}
40         ;
41
42 instr   : ESTE          { $$.dx =  1; $$.dy =  0; }
43         | NORTE         { $$.dy =  0; $$.dy =  1; }
44         | OESTE         { $$.dx = -1; $$.dy =  0; }
45         | SUR           { $$.dx =  0; $$.dy = -1; }
46         ;
47 %%
```

Del código de soporte, lo más importante son cuatro condicionales, cada una representa alguno de los tokens que simbolizan una dirección, además está comienza, y si no entra en ninguno de ellos, regresa TERMINA para que imprima las coordenadas nuevas del robot. El cual también sale del programa.

```
if (c == 'C')
      return COMIENZA;
else if (c == 'E')
      return ESTE;
else if (c == 'N')
      return NORTE;
else if (c == 'O')
      return OESTE;
else if (c == 'S')
      return SUR;
```

```
yosafat30@yosafat30-VirtualBox:~/Documentos/Compiladores/Practica 1$ ./robot
C EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
Nuevas coordenadas: 74,0
```

```
yosafat30@yosafat30-VirtualBox:~/Documentos/Compiladores/Practica 1$ ./robot
C NN EE
Nuevas coordenadas: 2,2
```

```
yosafat30@yosafat30-VirtualBox:~/Documentos/Compiladores/Practica 1$ ./robot
C SSSSSS OOOOOO
Nuevas coordenadas: -6,-6
```

```
yosafat30@yosafat30-VirtualBox:~/Documentos/Compiladores/Practica 1$ ./robot
SSSSS C
./robot: syntax error. Cerca de la linea 1
```

```
yosafat30@yosafat30-VirtualBox:~/Documentos/Compiladores/Practica 1$ ./robot
ESTO DEBE SER INVALIDO
./robot: syntax error. Cerca de la linea 1
```

Conclusiones

El uso de YACC hace que las gramáticas de los compiladores se puedan escribir de una forma elegante, a mí me gustó mucho cómo se escriben las reglas de la gramática y las acciones gramaticales, además, el hecho de que sea en C nos permite una gran libertad en todo lo que podemos hacer. Aunque, siempre he preferido Java la mayoría del tiempo, junto con Kotlin, sin duda para este caso, C es lo mejor que podemos escoger, es tan versátil que podemos hacer un sinfín de cosas si tenemos las reglas y acciones adecuadas. Me gustó mucho que tuve que buscar de varios lugares el cómo funciona por dentro algunas cosas de YACC,

aunque fue tardado, fue muy interesante y agradable ver que cada vez estaba más cerca de terminarlo.