



PROYECTO COMPILADOR LENGUAJE C

JOSE V. MARTI

58 GIIN – ESTRATEGIAS ALGORITMICAS
UNIVERSIDAD INTERNACIONAL DE VALENCIA
2023 - 2024

La realización del desarrollo de un compilador similar al lenguaje de programación C se ha llevado a cabo primero con la definición léxica de cada componente, seguido de la definición sintáctica y reglas semánticas y por último se ha comprobado su resultado ejecutando un fichero con varias instrucciones del lenguaje C. Todas estas fases se explican a continuación de una manera más detallada.

1.- Análisis Léxico.

Expresiones Regulares:

- DIGITO [0-9]+
- LETRAS [a-zA-Z]+
- CADENA "\"({LETRAS}|{DIGITO}|.)*\""
- AUMENTA "++"
- DISMINUYE "--"
- COMENTARIO "//".*"\n"

Símbolos:

"+", "-", "*", "/", "=", "!", "<=", ">=", ">", "<", "(", ")", "{", "}", "[", "]", "&", "||", ";", ":", ",", ".", "

Palabras Reservadas del Lenguaje C:

true, false, exit, int, float, char, printf, scanf, void
return, for, while, do, if, else, switch, case, break
default, main, include

2.- Análisis Sintáctico y Semántico.

Tokens:

DIGITO, LETRAS, CADENA, AUMENTA, DISMINUYE, COMENTARIO, PLUS, MINUS, MULT, DIV, EQUAL, NE, LE, GE, GT, LT, L_PAREN, R_PAREN, L_KEY, R_KEY, L_COR, R_COR, AND, OR, SEMIC, COMA, PUNTO, INT, FLOAT, CHAR, PRINTF, SCANF, FIN, VOID, RETURN, FOR, WHILE, DO, IF, ELSE, THEN, SWITCH, CASE, BREAK, DEFAULT, MAIN, INCLUDE, NUMBER, FLOAT_NUM, STRING, CHARACTER, ID, TRUE, FALSE

Reglas Semánticas:

programa: cabecera main L_PAREN R_PAREN L_KEY cuerpo return R_KEY;
(definición estructura del programa)

cabecera:

| cabecera INCLUDE ; (librerías a incluir)

main: tipoDato MAIN;

(función principal)

cuerpo:

cuerpo declaración	<i>(declaración de variables)</i>
cuerpo condiciones	<i>(estructuras condicionales)</i>
cuerpo bucles	<i>(estructuras recursivas)</i>
cuerpo salida	<i>(salida)</i>
cuerpo COMENTARIO;	<i>(comentarios)</i>

declaracion: tipoDato ID SEMIC

tipoDato ID EQUAL valor SEMIC
ID EQUAL valor SEMIC
ID AUMENTA SEMIC
ID DISMINUYE SEMIC;

condiciones: IF L_PAREN condicion R_PAREN L_KEY cuerpo R_KEY
 | IF L_PAREN condicion R_PAREN L_KEY cuerpo R_KEY ELSE
 L_KEY cuerpo R_KEY;

bucles: WHILE L_PAREN condicion R_PAREN L_KEY cuerpo R_KEY
 | DO L_KEY cuerpo R_KEY WHILE L_PAREN condicion R_PAREN SEMIC
 | FOR L_PAREN control_bucle SEMIC condicion SEMIC control_bucle
 R_PAREN L_KEY cuerpo R_KEY;

salida: PRINTF L_PAREN STRING R_PAREN SEMIC
 | PRINTF L_PAREN STRING COMA valor R_PAREN SEMIC; *(salida datos)*

condicion: valor EQUAL valor *(símbolo =)*
 | valor NE valor *(símbolo !=)*
 | valor GT valor *(símbolo >)*
 | valor GE valor *(símbolo >=)*
 | valor LT valor *(símbolo <)*
 | valor LE valor *(símbolo <=)*
 | condicion OR condicion
 | condicion AND condicion;

control_bucle: ID EQUAL valor
 | ID AUMENTA *(símbolo referente a i++)*
 | ID DISMINUYE; *(símbolo referente a i--)*

tipoDato: INT | DOUBLE | CHAR | VOID; *(definición tipo de datos)*

valor: NUMBER { \$\$=\$1; }
 | DOUBLE_NUM { \$\$=\$1; }
 | CHARACTER
 | ID
 | valor PLUS valor { \$\$ = \$1 + \$3; } *(operación suma)*
 | valor MINUS valor { \$\$ = \$1 - \$3; } *(operación resta)*
 | valor MULT valor { \$\$ = \$1 * \$3; } *(operación multiplicación)*
 | valor DIV valor { \$\$ = \$1 / \$3; } *(operación división)*

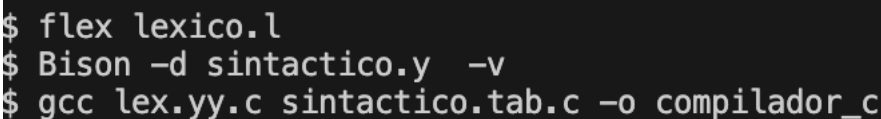
return: RETURN valor SEMIC | ; *(valor retorno en función y main)*

3.- Compilación Flex y Bison.

Una vez definidos tanto la parte léxica, como sintáctica y semántica de nuestro compilador, deberemos ahora compilar los diferentes componentes mediante los siguientes comandos:

- a) Compilación lexico.l: `flex lexico.l`
 - Genera el fichero: `lex.yy.c`
- b) Compilación sintactico.y: `Bison -d sintactico.y`
 - Genera los ficheros:
 - `sintactico.tab.h`
 - `sintactico.tab.c`

Completadas las fases de compilación, tanto del fichero léxico como del sintáctico, sin errores, finalmente faltará por compilar los ficheros `lex.yy.c` y `sintactico.tab.c` para generar el programa ejecutable, haciendo uso del comando: `gcc lex.yy.c sintactico.tab.c -o compilador_c`



```
$ flex lexico.l
$ Bison -d sintactico.y -v
$ gcc lex.yy.c sintactico.tab.c -o compilador_c
```

Ilustración 1 - Comandos ejecutados desde terminal.

4.- Pruebas Compilador.

Cuando hayamos terminado de compilar todos los procesos anteriores deberemos de comprobar que nuestro compilador funciona correctamente y para ello será necesario pasarle un fichero de texto que recoja todas las estructuras definidas en la parte léxica y sintáctica, las cuales son equivalentes al lenguaje C y ejecutar el programa `compilador_c`.

Dicho fichero de texto (test.txt) está compuesto de:

Librerías:

```
#include <stdio.h>
#include <locale.h>
#include <string.h>
```

Método Principal:

```
int main()
{

    //Declaración Variables
    int x = 5;
    int y = 2;
    double decimal = 546.79;
    printf("Valor Decimal: %f \n", decimal);
```

```
//Operaciones
int suma;
suma = 1 + 2;
printf("Suma: %d \n",suma);

int resta;
resta = 3 - 2;
printf("Resta: %d \n",resta);

int mult;
mult = 5 * 3;
printf("Multiplicación: %d \n",mult);

int div;
div = 8 / 2;
printf("División: %d \n",div);

Caracteres
char letra = 'C';
printf("Letra: %c \n", letra);

//Condiciones
if (x != y) {
    int operacion = x * y;
    printf("Resultado: %i \n",operacion);
}

if (x > y && x > 0) {
    printf("X Mayor a Y \n");
} else {
    printf("Resto Opciones \n");
}

//Bucles
int i = 1;
while (i <= 3) {
    printf("WHILE: %d \n", i);
    i++;
}

int inicio = 1;
do {
    printf("DO: %d \n", inicio);
    inicio++;
} while (inicio <= 3);

int valor;
for (valor = 1; valor < 3; valor++) {
    printf("FOR: %d \n", valor);
}

return 0; //retorno
}
```

Una vez hemos ejecutado nuestro compilador mediante el programa compilador_c, podemos observar como el proceso ha finalizado correctamente, por lo que la sintaxis y reglas utilizadas en el compilador han sido correctas.

```
$ ./compilador_c
Inicio Compilación
-----
Proceso Compilación finalizado Correctamente
-----
Fin Compilación
```

Ilustración 2 - Ejecución Compilador en terminal.

Si por ejemplo, añadiéramos a ese mismo fichero de texto, otra instrucción no definida como print en lugar de printf o el token sin definir @, el compilador devuelve un error de sintaxis y otro de carácter desconocido, respectivamente.

```
$ ./compilador_c
Inicio Compilación
-----
Error: syntax error
```

```
$ ./compilador_c
Inicio Compilación
-----
Error: "Caracter desconocido" en línea 1. Token = @
```

Ilustración 3 - Ejemplos errores forzados en compilador.

El mismo fichero de texto utilizado para comprobar el compilador lo convertimos a un programa de C y si lo ejecutamos podemos comprobar como las estructuras definidas devuelven los resultados esperados, demostrando así que nuestro compilador sigue las mismas reglas que el lenguaje de programación C.

```
$ gcc test.c -o test
$ ./test
Valor Decimal: 546.790000
Suma: 3
Resta: 1
Multiplicación: 15
División: 4
Letra: C
Resultado: 10
X Mayor a Y
WHILE: 1
WHILE: 2
WHILE: 3
DO: 1
DO: 2
DO: 3
FOR: 1
FOR: 2
```

Ilustración 4 - Estructuras del compilador mostradas en programa de C.