Universidad Nacional Autónoma de México

Facultad de Ingeniería

Ingeniería en Computación

Lenguajes Formales y Autómatas

Proyecto Final

Analizador léxico en flex y analizador sintáctico en yacc de las sentencias de control del lenguaje C

Grupo: 05

Integrante	Num. Cuenta
Hernández Rubio Dana Valeria	317345153
Soto Huerta Gustavo Isaac	318201458
Ugalde Santos Atzin	319057399

Profesora: Ing. Josefina Rosales García

Índice

Objetivo: Realizar el analizador léxico en flex y el analizador sintáctico en yacc de las sentencias de control del lenguaje C.

Desarrollo: Cada equipo analizará el lenguaje de las sentencias de control de forma generalizada. De ello realizará lo siguiente:

- ***Lista de Sentencias de Control en C***

Una sentencia de control se refiere a estructuras que controlan el flujo de ejecución del programa. Estas estructuras permiten tomar decisiones o repetir un bloque de código según ciertas condiciones.

- if

```
if\s(CONDICIÓN)\s{\n
        // ACCIÓN\n
}
```

- if-else

```
if\s(CONDICIÓN)\s{\n


        // ACCIÓN\n
}else{
        // ACCIÓN\n
}
```

- switch

```
switch\s(EXPRESIÓN )\s{
case\snúmero:
```

```
        // ACCIÓN\n
        break;
        // puede repetirse una o más veces
    default:
        // ACCIÓN\n
    }
```

- Bucles (loops)
  - while

```
    while\s(CONDICIÓN)\s{\n
        // ACCIÓN\n
    }
```

  - do-while

```
    do\s{\n
        // ACCIÓN\n
    }\swhile\s(CONDICIÓN);
```

  - for

```
    for\s(INICIALIZACIÓN;CONDICIÓN;ACTUALIZACIÓN)\s{
        // ACCIÓN\n
    }
```

- break

```
    break;
```

- continue

  Salta a la siguiente iteración en un bucle.

  continue;

- goto

  Permite saltar de un lugar a otro en el código de manera no lineal.

  goto\setiqueta;

➢ ***Expresiones regulares de cada una de las cadenas***

variable *{A-Za-z}+(_|{A-Za-z}|{0-9}*)*
número *{0-9}+\.{0-9}+*
cadena

if ***“if”[\s]*\([EXPRESIÓN][\s]*[COMPARACIÓN][EXPRESIÓN][\s]*\){\nACCIÓN\n}***

“if”[\s]*\([variable| número][\s]*[==|>|<|>=|<=|!=][variable| número][\s]*\){\n[^}]*\n}

if-else
“if”[\s]*\([EXPRESIÓN][\s]*[COMPARACIÓN][EXPRESIÓN][\s]*\){\nACCIÓN\n}“else”{\nACCIÓN\n}

“if”[\s]*\([variable| número][\s]*[==|>|<|>=|<=|!=][variable| número][\s]*\){\n[^}]*\n}“else”{\n[^}]*\n}

switch
“switch”[\s]*\([EXPRESIÓN]\)[\s]*{\n(?:[\s]*case[\s]*número:$[\s]*\n[^}]*\nbreak;$)+[\s]*default:$[\s]*\n[^}]*\n}
“switch”[\s]*\([variable| número]\)[\s]*{\n(?:[\s]*case[\s]*número:$[\s]*\n[^}]*\nbreak;$)+[\s]*default:$[\s]*\n[^}]*\n}

while **“while”[\s]*\([EXPRESIÓN][\s]*[RELACIONAL][EXPRESIÓN][\s]*\)[\s]*{\nACCIÓN\n}**

“while”[\s]*\([variable| número][\s]*[==|>|<|>=|<=|!=][variable| número][\s]*\)[\s]*{\n[^}]*\n}

do-while
“do”[\s]*{\nACCIÓN\n}[\s]*while][\s]*\([EXPRESIÓN][\s]*[RELACIONAL][EXPRESIÓN][\s]*\);$

“do”[\s]*{\n[^}]*\n}[\s]*while][\s]*\([variable| número][\s]*[==|>|<|>=|<=|!=][variable| número][\s]*\);$

for
“for”[\s]\([\s]*ASIGNACIÓN|DECLARACIÓN[\s]*;[\s]*[EXPRESIÓN][\s]*[RELACIONAL][EXPRESIÓN][\s]*;[\s]*ACTUALIZACIÓN[\s]*\){\nACCIÓN\n}*
“for”[\s]\([int | float | char | double][\s]*variable | variable[\s]*=[\s]*variable | número;[variable| número][\s]*[==|>|<|>=|<=|!=][variable| número][\s]*;[\s]*variable[\s]*++ | --\){\n[^}]*\n}*

break \t*break;$*
continue \t**continue**;$*
goto \t**goto**[\s]+\w+\s*;*

- ***Gramática de cada una de las sentencias***

- if

G_if(P, T, NT, S)
P:{ CONDICIÓN→ EXPRESIÓN COMPARACIÓN EXPRESIÓN
COMPARACIÓN→ **==|>|<|>=|<=|!=**
EXPRESIÓN→ variable | número
}
NT={CONDICIÓN, COMPARACIÓN , EXPRESIÓN }
T={if, (,), {, }, variable, operadores Relacionales, número}
S={CONDICIÓN}

- if-else

G_if-else(P, T, NT, S)
P:{
CONDICIÓN→ EXPRESIÓN COMPARACIÓN EXPRESIÓN
COMPARACIÓN→ **==|>|<|>=|<=|!=**
EXPRESIÓN→ variable | número
}
NT={CONDICIÓN, COMPARACIÓN , EXPRESIÓN }
T={if, else, (,), {, }, variable, operadores Relacionales, número}
S={CONDICIÓN}

- switch

G_switch_case(P, T, NT, S)
P:{
EXPRESIÓN→ variable | número
}

NT={EXPRESIÓN}
T={**switch**, (,), {, }, **case**, número, :, ; , **break**, **default**}
S={EXPRESIÓN}

- Bucles (loops)
  - while

G_while(P, T, NT, S)
P:{
CONDICIÓN→ EXPRESIÓN RELACIONAL EXPRESIÓN
EXPRESIÓN→ variable | número
RELACIONAL → == | != | < | > | <= | >=
}
NT={CONDICIÓN, EXPRESIÓN, RELACIONAL}
T={while, (,), {, }, variable, número}
S={CONDICIÓN}

  - do-while

G_do_while(P, T, NT, S)
P:{
CONDICIÓN→ EXPRESIÓN RELACIONAL EXPRESIÓN
EXPRESIÓN→ variable | número
RELACIONAL → == | != | < | > | <= | >=
}
NT={CONDICIÓN, EXPRESIÓN, RELACIONAL}
T={do, while, (,), {, }, variable, número}
S={CONDICIÓN}

- for

G_for(P, T, NT, S)
P:{
INICIALIZACIÓN → ASIGNACIÓN | DECLARACIÓN
ASIGNACIÓN → variable = EXPRESIÓN
DECLARACIÓN → TIPO variable
CONDICIÓN→ EXPRESIÓN COMPARACIÓN EXPRESIÓN
ACTUALIZACIÓN → variable ARITMÉTICOS
EXPRESIÓN→ variable | número
RELACIONAL → == | != | < | > | <= | >=
ARITMÉTICOS → ++ | --
TIPO → int | float | char | double
}

NT={INICIALIZACIÓN, ASIGNACIÓN, DECLARACIÓN, CONDICIÓN, ACTUALIZACIÓN, EXPRESIÓN, RELACIONAL, TIPO}
T={for, (,), {, }, variable, número, =, int, float, char}
S={CONDICIÓN}

- break

G_break(P, T, NT, S)
P:{
BREAK → break;
}
NT={BREAK}
T={break}
S={BREAK }

- continue

G_break(P, T, NT, S)
P:{
CONTINUE→ continue;
}

NT={CONTINUE}
T={continue}
S={CONTINUE }

- goto

G_goto(P, T, NT, S)
P:{
GOTO → goto etiqueta;
}
NT={GOTO}
T={goto, etiqueta}
S={GOTO}

➢ ***Obtener la tabla de clases :***

Clases	
0	Palabras Reservadas
1	Operadores de asignación
2	Operadores Lógicos
3	Símbolos Especiales
4	Números
5	Variables
6	Cadenas

➢ ***Obtener la tabla para cada clase:***

Palabras Reservadas

Palabras Reservadas	
0	for

1	if
2	switch
3	while
4	break
5	continue
6	goto
7	auto
8	case
9	char
10	const
11	default
12	do
13	double
14	else
15	enum
16	extern
17	int
18	float
19	short
20	long
21	return
22	signed

23	unsigned
24	sizeof
25	static
26	struct
27	void

Símbolos Especiales

Símbolos Especiales	
0	,
1	.
2	->
3	*
4	&
5	;
6	:
7	(
8	)
9	{
10	}
11	[
12	]

Operadores

Operadores Lógicos	
0	&&
1	||
2	!

Operadores Aritméticos	
0	+
1	-
2	*
3	/
4	++
5	--
6	%
7	=

Operadores Relacionales	
0	==
1	>
2	<
3	>=
4	<=
5	!=

Operadores Asignación	
0	+=
1	-=
2	*=
3	/=
4	%=

Dígitos

Dígitos	
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9

Variables

Variables

0	*i*

Cadenas

Cadenas	
0	*"Hola mundo"*

Conclusiones:

Hernández Rubio Dana Valeria:

En nuestro proyecto, hicimos el análisis de las sentencias de control en C, construyendo un sistema que pudiera entenderlas. Comenzamos por entender a fondo las estructuras como el "if", "if-else" , "switch", entre otras, y luego creamos reglas claras para describirlas. Las reglas, convertidas en expresiones regulares fueron la base para nuestro analizador léxico. Este, pudo identificar los patrones en el código.
Luego, pasamos a construir el analizador sintáctico, guiados por las reglas gramaticales que desarrollamos. Configuramos acciones que ayudaron a entender y organizar las sentencias de control correctamente. Esta parte no solo demostró nuestra comprensión de las estructuras del lenguaje C, sino también nuestra habilidad para llevar esa comprensión a la práctica.
En conclusión logramos hacer un analizador que puede procesar sentencias de control en C. Por lo que pudimos aplicar conceptos teóricos en un proyecto real. La colaboración entre los integrantes, la claridad en las reglas y la precisión en la implementación muestran el éxito del proyecto.

Soto Huerta Gustavo Isaac:

Lo primero que implementamos a la hora de hacer nuestro proyecto fue meternos de lleno en entender las sentencias de control en C. Al principio, desglosamos cómo lucen estas sentencias, desde la condición hasta el cuerpo. Luego, creamos reglas para describir cada una, y las tradujimos en expresiones regulares para que nuestro programa pudiera reconocerlas fácilmente. En cuanto al analizador léxico, el código en Flex refleja nuestro entendimiento de cómo deben identificarse y clasificarse los tokens en el código. La incorporación de clases específicas, como palabras

reservadas, operadores y variables, sirvió para la adaptabilidad del sistema a diferentes contextos de código.
El analizador sintáctico, implementado en Yacc, utilizó las reglas gramaticales establecidas para estructurar y analizar las sentencias de control. Este proceso fue gracias a nuestra comprensión de la teoría aprendida en clase y aplicada en este proyecto.

Ugalde Santos Atzin:

Para llevar a cabo el proyecto, aprovechamos la investigación previa sobre Flex y Yacc, las dos herramientas fundamentales para la creación de nuestro analizador léxico y sintáctico.
Con Lex, desarrollamos un analizador léxico que actúa como el primer filtro del código. Esta herramienta identifica patrones específicos, categorizando palabras clave, operadores y variables de manera estructurada. En esencia, Lex establece la base para el análisis sintáctico al organizar y etiquetar las diversas partes del código. Por otro lado, Yacc toma las piezas identificadas por Lex y les da sentido en términos de gramáticas definidas.
Este proceso se llevó a cabo gracias al análisis detallado de las sentencias de control en C, entendiendo cada componente, desde las condiciones hasta las acciones. Creamos gramáticas y expresiones regulares específicas para cada sentencia, definiendo la estructura de estas en términos de expresiones, comparaciones y acciones. Estas reglas, expresadas en códigos gramaticales, se transformaron posteriormente en expresiones regulares que guiaron nuestro analizador léxico.

Bibliografía:

1. Aitken, P. G., & Jones, B. (1994). *Aprendiendo C en 21 días*. Sams.
2. Deitel, H. M., & Deitel, P. J. (2004). *Cómo programar en C/C ++ y Java*. Pearson Educación.
3. Ammeraal, L. (1991). *C for programmers: A Complete Tutorial Based on the ANSI Standard*.

4. *Mi primer proyecto utilizando Yacc y Lex*. (2020, 1 octubre). Erick Navarro. https://ericknavarro.io/2020/10/01/27-Mi-primer-proyecto-utilizando-Yacc-y-Lex/