UNIVERSIDAD DE GRANADA

Escuela Tecnica Superior de Ingenieria
Informatica y Telecomunicaciones

Practicas Modelos de Computación

Grupo B3

Juan Luis Torres Ramos

15 Enero 2024

Índice

1 Practica 1

Encuentra una gramática libre del contexto para generar cada uno de los siguientes lenguajes:

1. $L = \{a^i b^j \mid i, j \in \mathbb{N},\ i \leq j\}$.
2. $L = \{a^i b^j a^j b^i \mid i, j \in \mathbb{N}\}$.
3. $L = \{a^i b^i a^j b^j \mid i, j \in \mathbb{N}\}$.
4. $L = \{a_i b_i \mid i \in \mathbb{N}\} \cup \{b_i a_i \mid i \in \mathbb{N}\}$.
5. $L = \{uu^{-1} \mid u \in \{a, b\}^*\}$.
6. $L = \{a^i b^j c^{i+j} \mid i, j \in \mathbb{N}\}$.

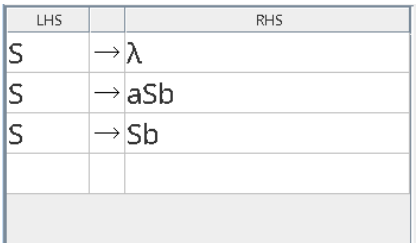
donde $\mathbb{N}$ es el conjunto de los numeros naturales incluyendo el 0

Pasos para resolver el ejercicio:

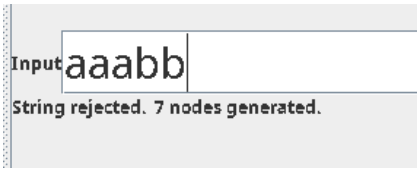
1. Determinar los símbolos terminales y no terminales.
2. Determinar el símbolo inicial.
3. Analizar el lenguaje para determinar qué se pide.
4. Determinar las reglas de producción.
5. Comprobar con JFLAP

1.1 $L = \{a^i b^j \mid i, j \in \mathbb{N},\ i \leq j\}$.

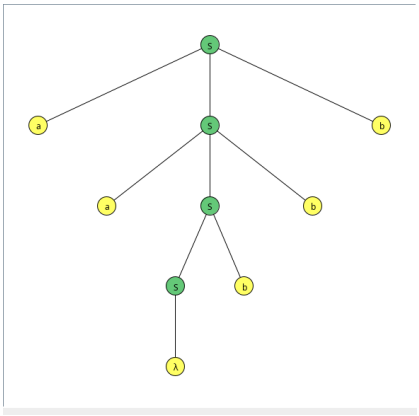
1. Los símbolos terminales serán $\{a, b\}$ y los simbolos no terminales serán S y B.
2. El símbolo inicial será S.
3. Analizar el lenguaje para determinar qué se pide. En este caso, se pide que la cadena tenga un número de a menor o igual que el número de b. Por ejemplo, *aabbb* y *aabb* pertenecen al lenguaje, pero *aab* no.
4. Determino las reglas de producción:
   - $S \to \epsilon$ (genero la cadena vacía).
   - $S \to aSb$.
   - $S \to Sb$.
5. compruebo con JFLAP que la gramática es correcta.



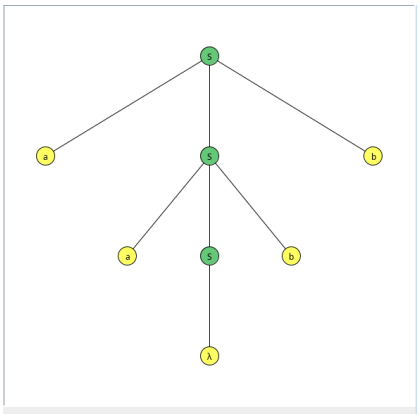
(a) la producción



(b) la cadena *aaabb*



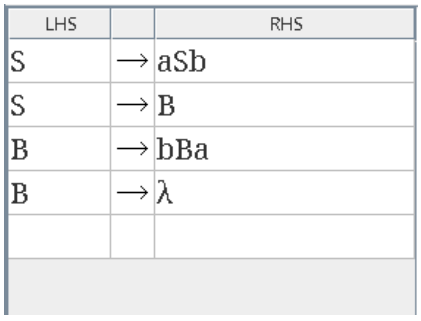
(c) la cadena *aabbb*



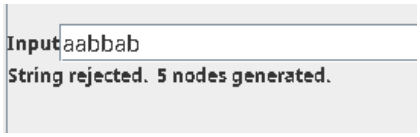
(d) la cadena *aabb*

1.2 $L = \{a^i b^j a^j b^i \mid i, j \in \mathbb{N}\}$.

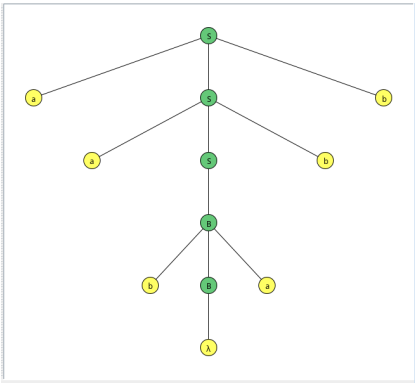
1. Los símbolos terminales serán $\{a, b\}$ y los simbolos no terminales serán S y B.
2. El símbolo inicial será S.
3. El lenguaje nos pide generar una cadena de 4 caracteres donde primero se generen $a^i b^j$ y luego $a^j b^i$, es decir en los extremos un numero caracteres i y en los caracteres del centro un numero de caracteres j. Por ejemplo, *aababb* y *ab* pertenecen al lenguaje, pero *aabbab* no.
4. Determino las reglas de producción:
   - $S \to aSb$ (genero mismo numero de caracteres en los extremos).
   - $S \to B$.
   - $B \to bBa$ (genero mismo numero de caracteres en el centro).
   - $B \to \epsilon$ (genero la cadena vacía).
5. compruebo con JFLAP que la gramática es correcta.



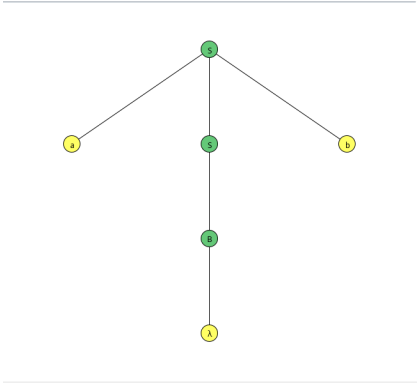
(a) la producción



(b) la cadena *aabbab*



(c) la cadena *aababb*



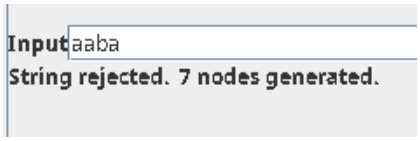
(d) la cadena *ab*

1.3 $L = \{a^i b^i a^j b^j \mid i, j \in \mathbb{N}\}$.

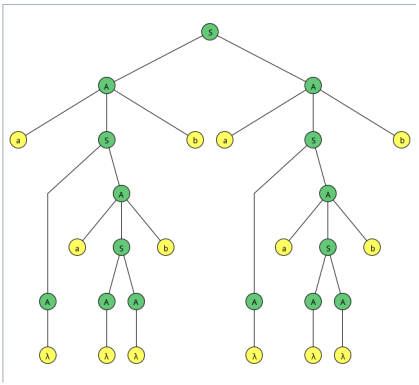
1. Los símbolos terminales serán $\{a, b\}$ y los simbolos no terminales serán S y B.
2. El símbolo inicial será S.
3. El lenguaje nos pide generar cadenas de 4 caracteres de la forma *abab* donde los dos primeros caracteres tengan el mismo numero de caracteres y para los dos ultimos caracteres tambien tengan la misma cantidad. Ejemplos de cadenas serían *aabbaabb* ,*aabbab* pero no acepta *aaba*
4. Determino las reglas de producción:
   - $S \to AA$ (simbolo inicial).
   - $A \to aSb$. (genero $\{a^i b^i | i \in \mathbb{N}\}$).
   - $A \to \epsilon$ (genero la cadena vacía).
5. compruebo con JFLAP que la gramática es correcta.

LHS		RHS
S	⟶	AA
A	⟶	aSb
A	⟶	λ

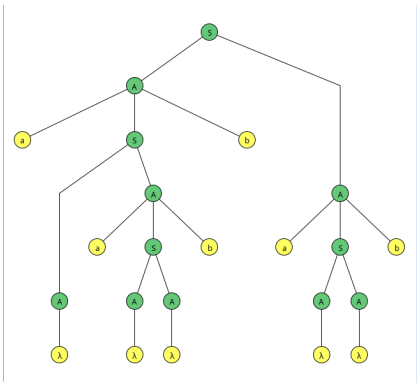
(a) la producción



(b) la cadena *aaba*



(c) la cadena *aabbaabb*



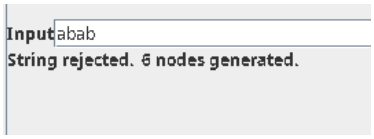
(d) la cadena *aabbab*

1.4 $L = \{a_i b_i \mid i \in \mathbb{N}\} \cup \{b_i a_i \mid i \in \mathbb{N}\}$.

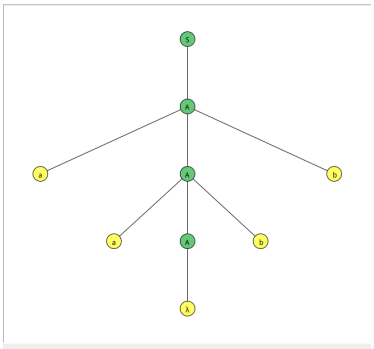
1. Los símbolos terminales serán $\{a, b\}$ y los simbolos no terminales serán S , A B.
2. El símbolo inicial será S .
3. Combina dos conjuntos de cadenas: el primero contiene cadenas de la forma $\{a_i b_i \mid i \in \mathbb{N}\}$, y el segundo contiene cadenas de la forma $\{b_i a_i \mid i \in \mathbb{N}\}$.Las cadenas *aabb bbaa* lo cumplen mientras *abab* no lo cumple Lo resolvemos por partes
4. Determino las reglas de producción:
   - Podemos generar $\{a_i b_i \mid i \in \mathbb{N}\}$.
     $A \to aAb$, $A \to \epsilon$.
   - Por otro lado $\{b_i a_i \mid i \in \mathbb{N}\}$.
     $B \to bBa$, $B \to \epsilon$.
   - El lenguaje L se puede generar añadiendo .
     $S \to A$, $S \to B$.
5. compruebo con JFLAP que la gramática es correcta.

LHS		RHS
S	→	A
S	→	B
A	→	aAb
A	→	λ
B	→	bBa
B	→	λ

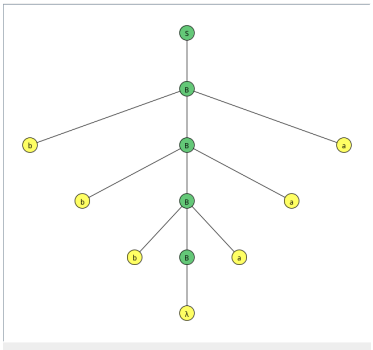
(a) la producción



(b) la cadena *abab*



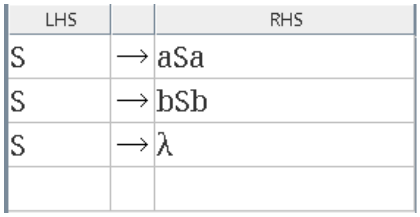
(c) la cadena *aabb*



(d) la cadena *bbbaaa*

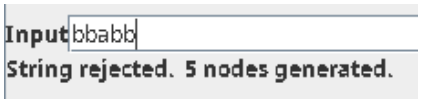
1.5 $L = \{uu^{-1} \mid u \in \{a, b\}^*\}$.

1. Los símbolos terminales serán $\{a, b\}$ y los simbolos no terminales serán S.
2. El símbolo inicial será S.
3. Analizar el lenguaje para determinar qué se pide. En este caso,se pide generar cadenas que son palíndromos formados por caracteres 'a' y 'b'. Cadenas que pertenecen al lenguaje son *abba* y *bbaabb* pero no *bbabb*.
4. Determino las reglas de producción:
   - $S \to \epsilon$ (genero la cadena vacía).
   - $S \to aSa$.
   - $S \to bSb$.
5. compruebo con JFLAP que la gramática es correcta.

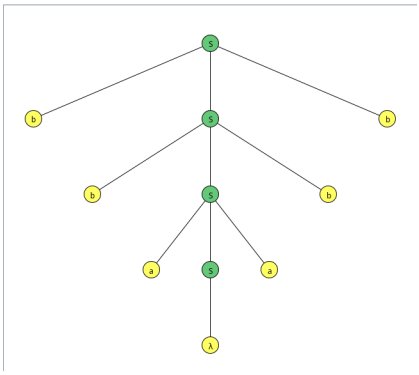


LHS		RHS
S	⟶	aSa
S	⟶	bSb
S	⟶	λ

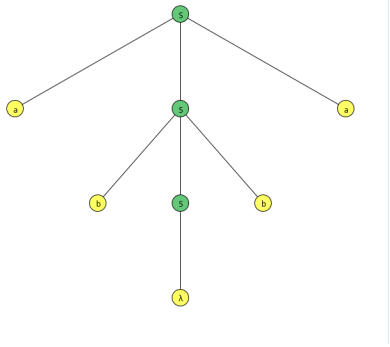
(a) la producción



(b) la cadena *bbab*



(c) la cadena *bbaabb*



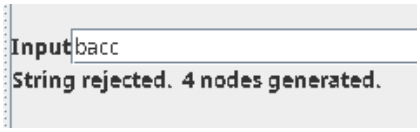
(d) la cadena *abba*

1.6 $L = \{a^i b^j c^{i+j} \mid i, j \in \mathbb{N}\}$.

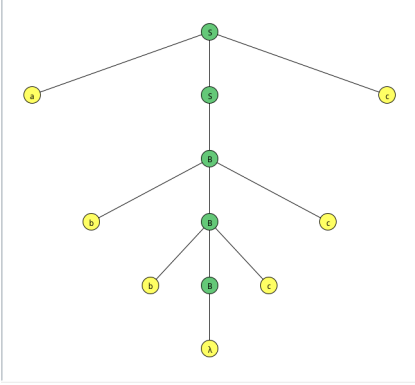
1. Los símbolos terminales serán $\{a, b, c\}$ y los simbolos no terminales serán S.
2. El símbolo inicial será S.
3. En este caso,se pide generar cadenas donde la cantidad de 'a's y 'b's es igual y la cantidad total de 'c's es la suma de las cantidades de 'a' y 'b' . Cadenas que cumplen la gramatica son *abbccc* y *aaabcccc* pero no *bacc*
4. Determino las reglas de producción:
   - $S \to aSc$ (genero la cadena vacía).
   - $S \to B$.
   - $B \to bBc$.
   - $B \to \epsilon$.
5. compruebo con JFLAP que la gramática es correcta.

LHS		RHS
S	⟶	aSc
S	⟶	B
B	⟶	bBc
B	⟶	λ

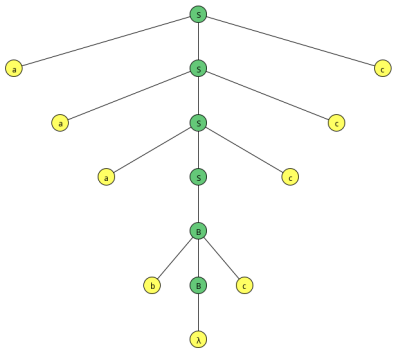
(a) la producción



(b) la cadena *bacc*



(c) la cadena *abbccc*



(d) la cadena *aaabcccc*

2 Practica 2

Analizadores léxicos, problemas de mineria, trabajo Lex.

2.1 Tareas a realizar

1. Formar un grupo de trabajo compuesto por una, dos o tres personas.
2. Cada grupo de trabajo debe pensar un problema original de procesamiento de textos. Para la resolución de este problema debe ser apropiado el uso de Lex, o sea, se debe resolver mediante el emparejamiento de cadenas con expresiones regulares y la asociación de acciones a cada emparejamiento.
3. Cada grupo debe resolver el problema propuesto usando Lex. Se deberá realizar una memoria donde se presente una descripción del problema y su solución, además de entregar electrónicamente los ficheros de texto con la implementación de la solución.
4. Esta práctica deberá ser entregada antes del día 15 de Enero de 2024. Se entregará a través de la plataforma PRADO en un fichero .zip conteniendo todos los archivos de esta práctica. Sólo es necesario que lo entregue uno de los componentes del grupo.

Pasos para resolver el ejercicio:

1. Descripcion del problema
2. Solucion
3. Codigo lex

2.2 Descripcion del Problema

Soy un nuevo profesor de la asignatura de Fundamentos de Programacion. Tras corregir varios ejercicios de los alumnos me he dado cuenta que la cantidad de comentarios explicando el codigo va relacionada con la nota del ejercicio. Por lo que he decidido crear un programa que calcule la densidad de comentarios en un codigo fuente en C para evaluar positivamente a los alumnos que comenten su codigo.

2.3 Densidad de comentarios codigo

Tu tarea es desarrollar un programa en Lex que calcule la densidad de comentarios en un código fuente en C. La densidad de comentarios se define como el porcentaje del código total que está ocupado por comentarios.

2.4 Pasos

El alumno ha entregado su ejercicio de C correspondiente de la asignatura, voy a calcular la densidad de comentarios con la siguiente formula:

$$\text{Densidad de comentarios} = \frac{\text{Total de letras en un comentario}}{\text{Total de letras en el codigo}}$$

1. Creo 2 variables globales, para contar letras en el código y en comentarios.
2. Defino 2 estados `INCOMMENTBLOCK` y `INCOMMENTLINE` para manejar por separado los dos tipos de comentarios en C: comentarios en línea y comentarios en bloque.
3. Defino una función `contarLetras` que cuenta únicamente letras; no cuenta espacios en blanco, tabuladores, saltos de línea ni retornos de carro.
4. Defino reglas de Flex para reconocer los comentarios:
   - Si encuentra "/*", comienza el subestado `INCOMMENTBLOCK` y termina con "*/".
   - Si encuentra "//", comienza el subestado `INCOMMENTLINE` y termina con un salto de línea.
   - Dentro del estado `INCOMMENTBLOCK`, selecciono para cualquier carácter que no sea un asterisco (para evitar contar el fin del comentario `*/`) ni un salto de línea. Para comentarios en línea, solo no cuento el salto de línea.
   - El texto seleccionado corresponde a la variable `yytext`, la cual introduzco en mi función `contarLetras`.
   - Imprimo cada comentario encontrado haciendo print a yytext, tambien indico tipo comentario y su longitud

- Para referirme a todo el código, uso `.|\n`, haciendo referencia a cualquier carácter y un salto de línea. Asi calculo la longitud total del fichero.

5. Por último, calculo la densidad de comentarios con la fórmula anterior.

Para ejecutar el programa usaremos un Makefile

```
all: comentador

comentador: lex.yy.c
    gcc -o comentador lex.yy.c -lfl

lex.yy.c: comentador.l
    flex comentador.l

run: comentador
    ./comentador < ./ejemplos/ejemplo1.c
    ./comentador < ./ejemplos/ejemplo2.c
```

(a) Makefile

```
$ make all
$ make run
```

(b) Ejemplo de ejecución

```
#include <stdio.h>
#include <stdlib.h>

// funcion de ejemplo
void imprimirMensaje() {
    printf("! Hola, mundo!\n");
}

/*
    comentarios
    multilinea
*/

int main{
    // Llamada a la funcion
    imprimirMensaje();
    return 0;
}
```

(c) Ejemplo1.c

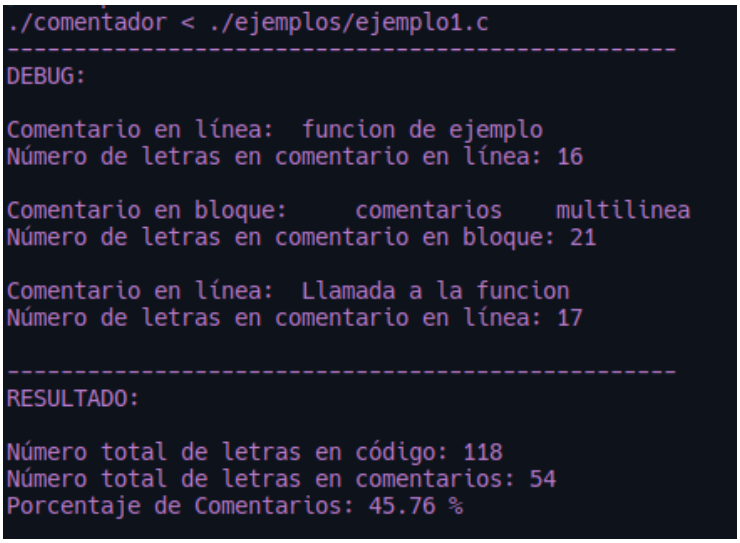
```c
// Pepe1
//  Pepe 1

#include <stdio.h>
#include <stdlib.h>

void imprimirMensaje() // paso1: imprime mensaje en pantalla
{
    printf("! Hola, mundo!\n");
}

/*
    ahora viene el main
*/


int main // paso2: llamo a / la funcion
{
    imprimirMensaje();
    return 0;
}
```

(d) Ejemplo2.c

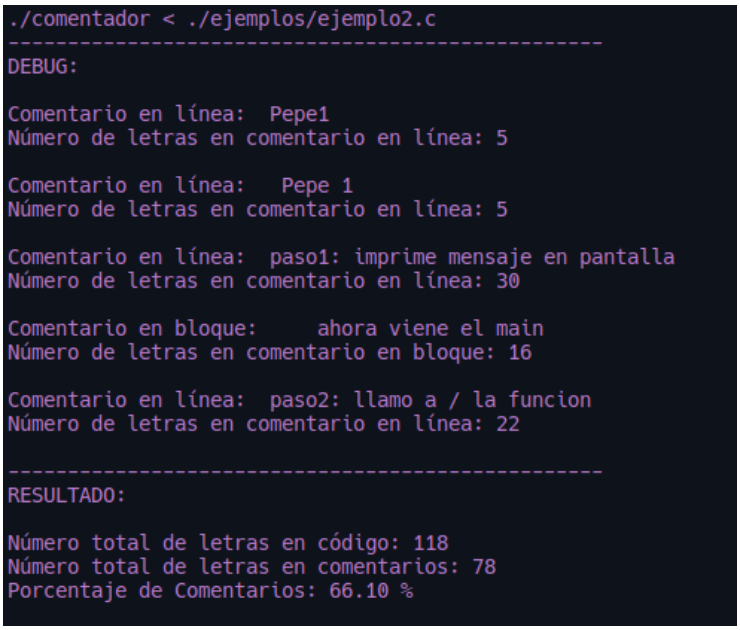
```
%{
#include <stdio.h>
#include <stdlib.h>

int total_letters = 0;
int comment_letters = 0;
int comment_letters_global = 0;
int contarLetras(const char* texto);
%}

%x INCOMMENTBLOCK INCOMMENTLINE

%%
"/*" {
    BEGIN(INCOMMENTBLOCK);
    printf("\nComentario en bloque: ");
    comment_letters = 0;
}

<INCOMMENTBLOCK>"*/" {
    BEGIN(INITIAL);
    printf("\nNumero de letras en comentario en bloque: %d\n",
comment_letters);
    comment_letters = 0;
}

<INCOMMENTBLOCK>[^*\n]+ {
    int letras_comentario = contarLetras(yytext);
    printf("%s", yytext);
    comment_letters += letras_comentario;
    comment_letters_global += letras_comentario;
}

```

```
33      <INCOMMENTBLOCK>\n {
34          // No contar el salto de linea
35      }
36
37      "//" {
38          BEGIN(INCOMMENTLINE);
39          printf("\nComentario en linea: ");
40      }
41
42      <INCOMMENTLINE>\n {
43          BEGIN(INITIAL);
44          printf("\nNumero de letras en comentario en linea: %d\n",
    comment_letters);
45          comment_letters = 0;
46      }
47
48      <INCOMMENTLINE>[^\n]+ {
49          printf("%s", yytext);
50          comment_letters += contarLetras(yytext);
51          comment_letters_global += contarLetras(yytext);
52      }
53
54      .|\n {
55          total_letters += contarLetras(yytext);
56      }
57      %%
58      // sin salto de linea al final, solo cuento letras simbolos y
    numeros
59      int contarLetras(const char* texto) {
60          int contador = 0;
61          while (*texto) {
62              if (*texto != '\n' && *texto != '\t' && *texto != ' '
    && *texto != '\r') {
63                  contador++;
64                  }
65                  texto++;
66              }
67          return contador;
68      }
69      // conslato de linea
70      int main(int argc, char* argv[]) {
71          printf("DEBUG:\n");
72          yylex();
73
74          printf("RESULTADO:\n");
75          printf("\nNumero total de letras en codigo: %d\n",
    total_letters);
76          printf("Numero total de letras en comentarios: %d\n",
    comment_letters_global );
77          printf("Porcentaje de Comentarios: %.2f \%\n\n", (float)(
    comment_letters_global) / total_letters * 100);
78          return EXIT_SUCCESS;
79      }
80
```

(e) Codigo Analizador Lex



(f) Resultado de la ejecución del programa



(g) Resultado de la ejecución del programa

Analisis Resultado

Podemos ver que el programa ha detectado los comentarios y diferenciado si es un comentario en linea o un comentario en bloque. Calcula correctamente tanto las letras de cada comentario como el total de letras en el codigo y por ultimo calcula el porcentaje de comentarios en el codigo correspondiente. Ahora el maestro, viendo el porcentaje de comentarios de cada ejercicio de FP, puede evaluar positivamente a los alumnos que comenten su codigo correctamente. Tampoco se puede abusar de los comentarios, ya que el maestro puede ver el porcentaje de comentarios y si es demasiado alto puede penalizar al alumno.

3 Practica 3

Diseña una máquinas de estados finitos, en particular la máquina de Mealy, para simular la codificación y decodificación del código Enigma. Implementa un conjunto de estados y transiciones que reflejen el proceso de cifrado y descifrado característico del Enigma. Utiliza JFLAP para simular y visualizar la máquina de Mealy que actúa como codificador y decodificador.

3.1 Maquina Enigma

Durante la Segunda Guerra Mundial, la máquina Enigma era utilizada para encifrar las comunicaciones militares de Alemania. La Enigma, creada por Arthur Scherbius y adoptada por el ejército alemán en los años 30, constaba de un teclado y rotores que se podían girar.

Los rotores, que eran discos rotatorios con letras del alfabeto conectados eléctricamente en cascada, eran la fuente de la complejidad. Antes de cifrar un mensaje, se configuraban los rotores para determinar la sustitución de letras.

Un solo "reflector" reflejaba la señal a través de los rotores, lo que aumentaba la complejidad del cifrado. Además, después de cada pulsación, los rotores giraban, cambiando la configuración y dificultando los intentos de descifrado.

A pesar de la fuerza de la Enigma, los aliados, especialmente aquellos en Bletchley Park, lograron descifrar las comunicaciones encriptadas, lo que ayudó a las fuerzas del Eje a ganar.

(a) Maquina Enigma

3.2 Reglas

Debido a la complejidad de la máquina Enigma, he decidido mostrar las reglas básicas de reemplazo de letras en uno de sus rotores. Utilizo las entradas binarias 0 y 1 y sus respectivas salidas para mapear el rotor en mi representación.

Cada uno de los tres nodos de estado en el diagrama se llama A, B y C, y representa una configuración particular del rotor. Estos estados podrían referirse a las primeras ubicaciones de los rotores de la máquina Enigma.

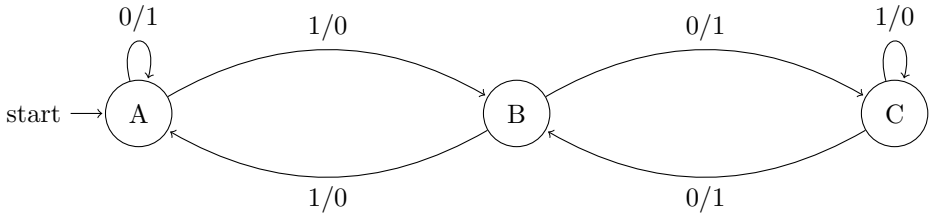
Cada transición se representa con una flecha. Por ejemplo, una flecha de A a B está marcada con "0/1", lo que significa que si la entrada es 0, la salida será 1.

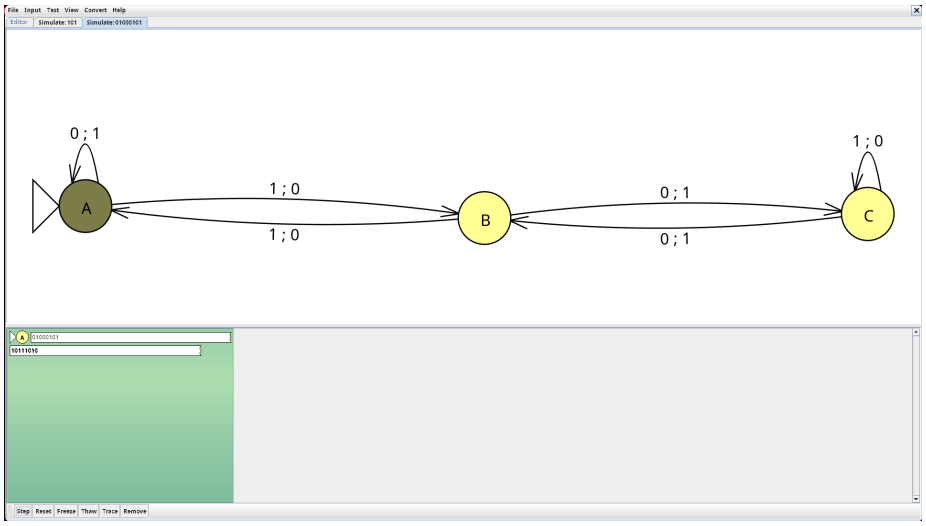
Reglas que he definido:

- De A a B: "1 / 1"
- De A a B: "1 / 0"
- De B a C: "0 / 1"
- De B a A: "1 / 0"
- De C a C: "1 / 0"
- De C a B: "0 / 1"

3.3 Codificador

- Entrada Binaria (0 o 1): Recibe una entrada binaria, por ejemplo, '01000101'.
- Transición a través de los Rotores: Aplica las reglas de sustitución del rotor correspondiente. Siguiendo el ejemplo, podría cambiar '10' a '11' según las reglas definidas.
- Salida Cifrada: La salida es la representación cifrada de la entrada original. En nuestro ejemplo, la salida sería '10111010'.



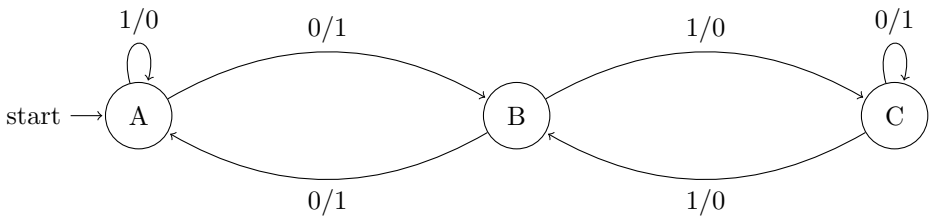


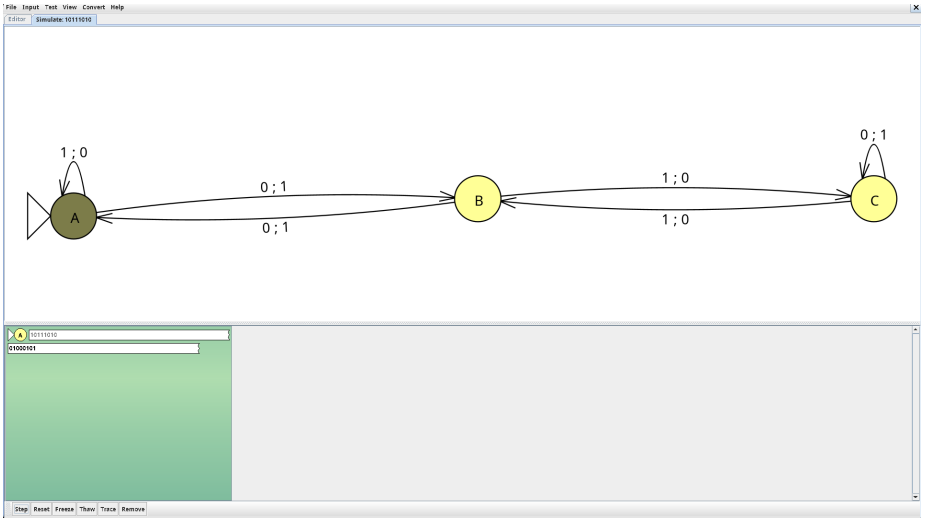
(b) codificador

3.4 Decodificador

Un decodificador es, en esencia, una máquina de estados finitos que hace lo que un codificador hace. El codificador de tu caso tenía entradas y salidas, y el decodificador debe tener entradas y salidas para revertir ese proceso.

- Entrada Binaria (0 o 1): Recibe la entrada binaria de salida del codificador, por ejemplo, '10111010'.
- Transición a través de los Rotores: Aplica las reglas de sustitución del rotor correspondiente, pero esta vez a la inversa.
- Salida Cifrada: La salida es la salida original luego de cifrarlo. En nuestro ejemplo, la salida sería '01000101'.





(c) decodificador