**INSTITUTO POLITÉCNICO NACIONAL**

**ESCUELA SUPERIOR DE CÓMPUTO**

**ZEPEDA FLORES ALEJANDRO DE JESÚS**

**NO. BOLETA 2016601853**

**TEORÍA COMPUTACIONAL**

**PROF. LUZ MARÍA SANCHÉZ GARCÍA**

**20 DE ABRIL DE 2018**

**INTRODUCCIÓN**

Las gramáticas de tipo 2 o gramáticas independientes del contexto, son las que generan los lenguajes libres o independientes del contexto. Los lenguajes libres del contexto son aquellos que pueden ser reconocidos por un autómata de pila determinístico o no determinístico.

* Una gramática libre de contexto (GLC), describe un lenguaje libre de contexto.
* Son útiles para describir bloques anidados en lenguajes de programación ya que describen su sintaxis.
* Son llamadas así porque el elemento no terminal del lado derecho se puede sustituir sin importar el contexto en que este.
* Su característica es que piden que solamente exista un no terminal el lado izquierdo de la producción.

**PLANTEAMIENTO DEL PROBLEMA**

Realización de un programa que reconozca un Lenguaje libre de contexto predeterminado (LLC). El programa deberá leer desde un archivo o desde el teclado una GLC G y derivado de él reconocer las cadenas del lenguaje LLC.

• El programa debe leer:

* G = ( VN, VT, S, δ ) donde:
* VN: Los No Terminales a usar en la gramática. (S, A, B, etc.)
* VT: Los Terminales (a, b, etc.)
* S: El símbolo inicial (S)
* δ: Las reglas de producción (S->aA, A-> bB, B->b, etc.)

Se pide mostrar paso a paso la derivación de la cadena producto de sus producciones o el árbol de derivación.

**DIAGRAMA**

Indagar acerca de las reglas a seguir para poder generarlas.

Plantear la solución al problema de generar una gramática independiente libre de contexto.

Existen errores

Verificar que al final del árbol de derivación, se haya generado la gramática solicitada.

FIN

Elaborar reporte.

NO

SI

Corregir errores y realizar pruebas.

Realizar las pruebas necesarias para checar la implementación.

Crear el árbol de derivación.

Seleccionar una gramática independiente de contexto para implementar.

INICIO

**IMPLEMENTACIÓN DE LA SOLUCIÓN**

Para hacer este programa más general, realizamos la lectura de la expresión desde un archivo, además, para el usuario es más sencillo escribir en un documento de texto que en la terminal.

char \* readfile(char \* namefile){

int posicion = 0, tam = 0, i = 0;

char caracter; char \* lenguaje;

FILE \* file = fopen(namefile,"r"); //Creación del apuntador

fflush(stdin); caracter = fgetc(file);

while(caracter != EOF){ //Lectura del archivo

tam = tam + 1; //Longitud de la expresión

caracter = fgetc(file);

}

fclose(file); fopen(namefile,"r");

lenguaje = (char \*)malloc(sizeof(char)\*(tam+1));

fflush(stdin); caracter = fgetc(file);

for(i=0; i<tam; i++){

lenguaje[posicion++] = caracter; //Copia de la expresión

caracter = fgetc(file);

}

lenguaje[posicion] = '\0';

fclose(file);

return lenguaje;

}

Con esta función, verificamos que la expresión se encuentre bien escrita. Posteriormente, procedemos a realizar el árbol de derivación y a formar la cadena terminal; en caso de que no se genere cadena terminal, imprimimos épsilon.

int verificar(char \* cadena, int repeat){

char s; char a; char b; char x = 0;

int i = 0, j = 0, l = 0, tam = 0;

i = tam = strlen(cadena)-1;

while(cadena[i]!=32){//Recorremos la cadena hasta un espacio

if(cadena[i]>=65 && cadena[i]<=90){ // Verificamos el cont

s = cadena[i]; break;

}

else

l = l+1;

i = i-1;

} j = tam-l-5;

b = cadena[tam]; a = cadena[tam-l-1];//Obtenemos la expresión

for(x=1; x<=repeat ;x++){ //Impresión del árbol de derivación

for(i=0; i<j\*x ;i++)

printf("%c",a);

printf("%c",s);

for(i=0; i<l\*x ;i++)

printf("%c",b);

printf("\n");

}

return 0;

}

Ahora trabajaremos un poco con a interfaz del programa, cumpliendo con los requerimientos del mismo, pero la a vez tratando de hacer más agradable para el usuario la interacción con el programa.

Como primer paso solicitamos al usuario la longitud de la cadena que desea validar, después empezamos las validaciones con las funciones previamente programadas.

La lectura del archivo la realizamos nosotros mismos para evitar que el usuario cometa un error y agregue el archivo incorrecto.

int main(int argc, char \*argv[]) {

int cantidad = 0, repeat = 0;

struct informacion dato;

do{

printf("Practica 4 - Morelos Ordonez Pedro Luis\n");

printf(" Zepeda Flores Alejandro de Jesús\n\n");

printf("Cuantas cadenas deseas imprimir? ");

scanf("%d", &cantidad);

verificar(readfile("archivo.txt"),cantidad);

printf("\n\nDesea repetir el programa? Si (1) NO (2) ");

scanf("%d",&repeat);

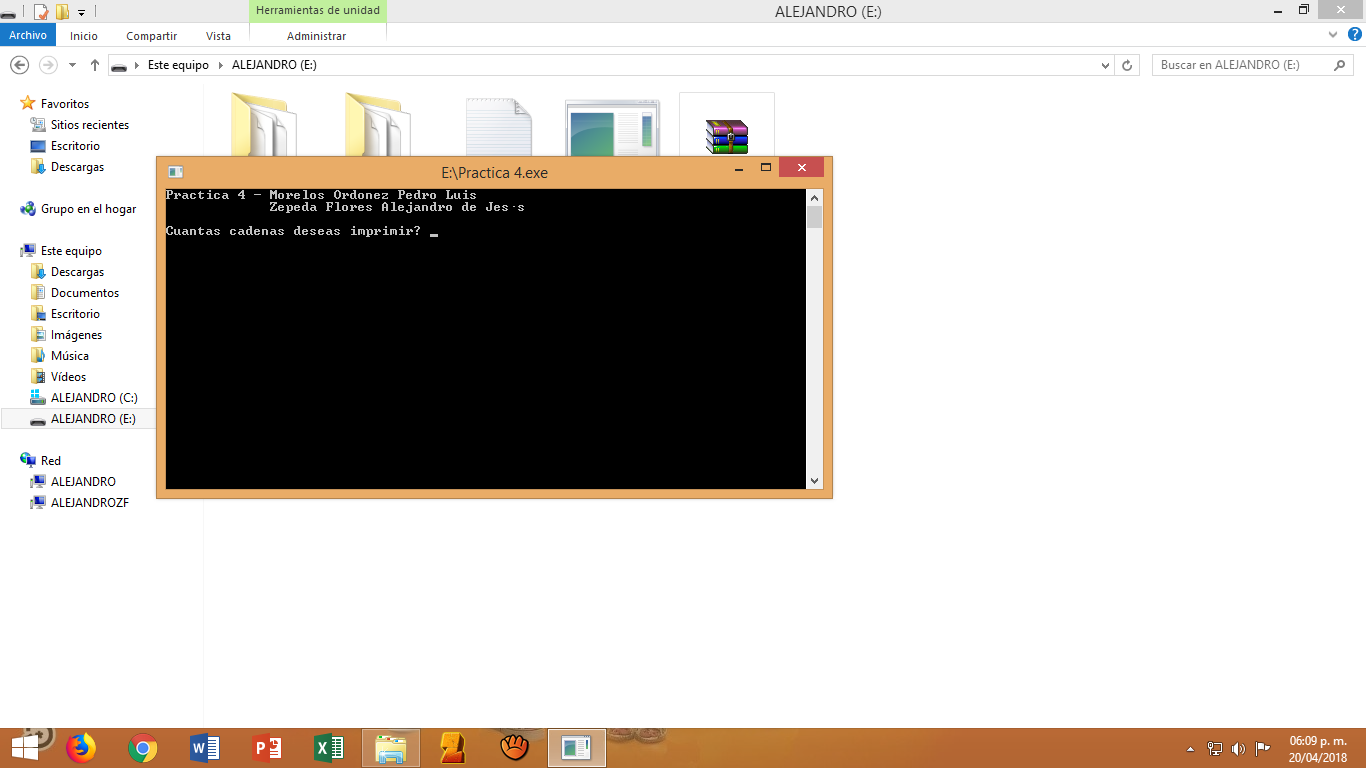
system("cls");

}while(repeat == 1);

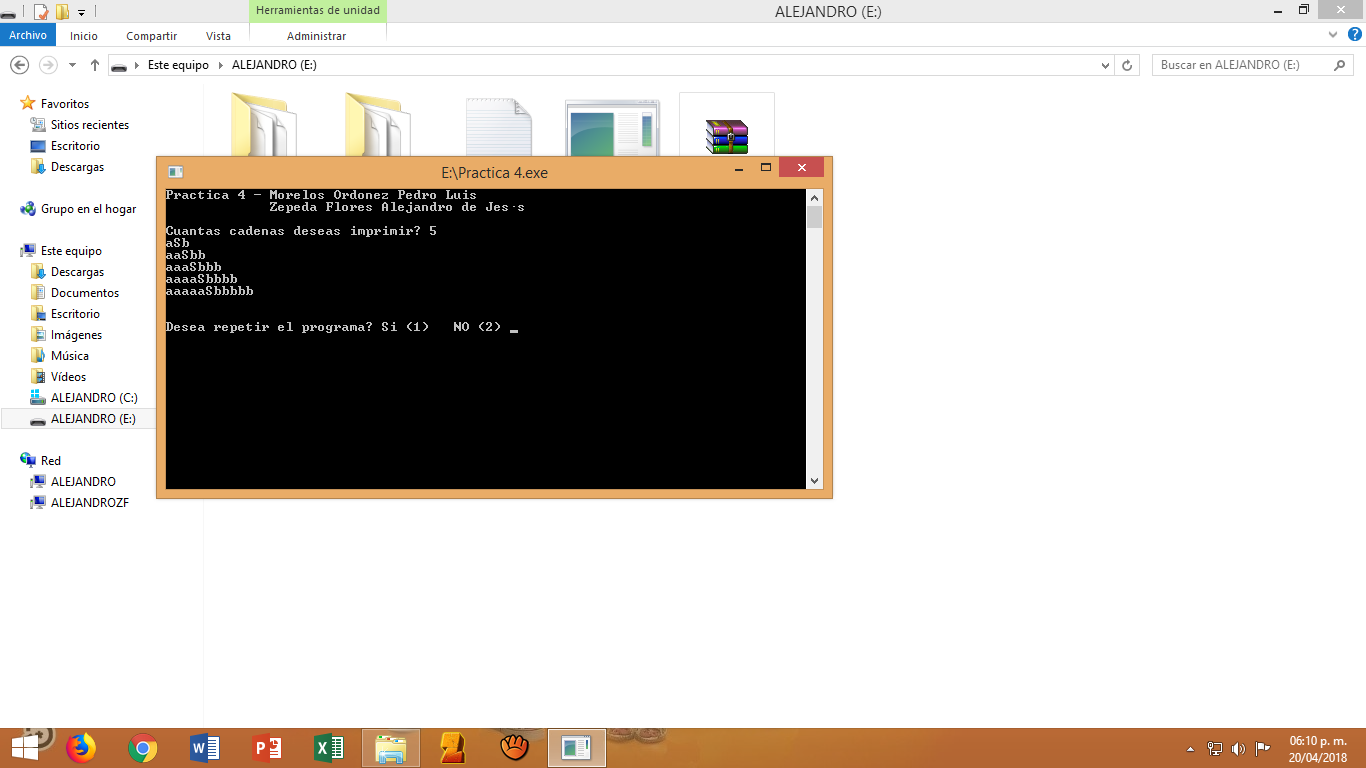
return 0;

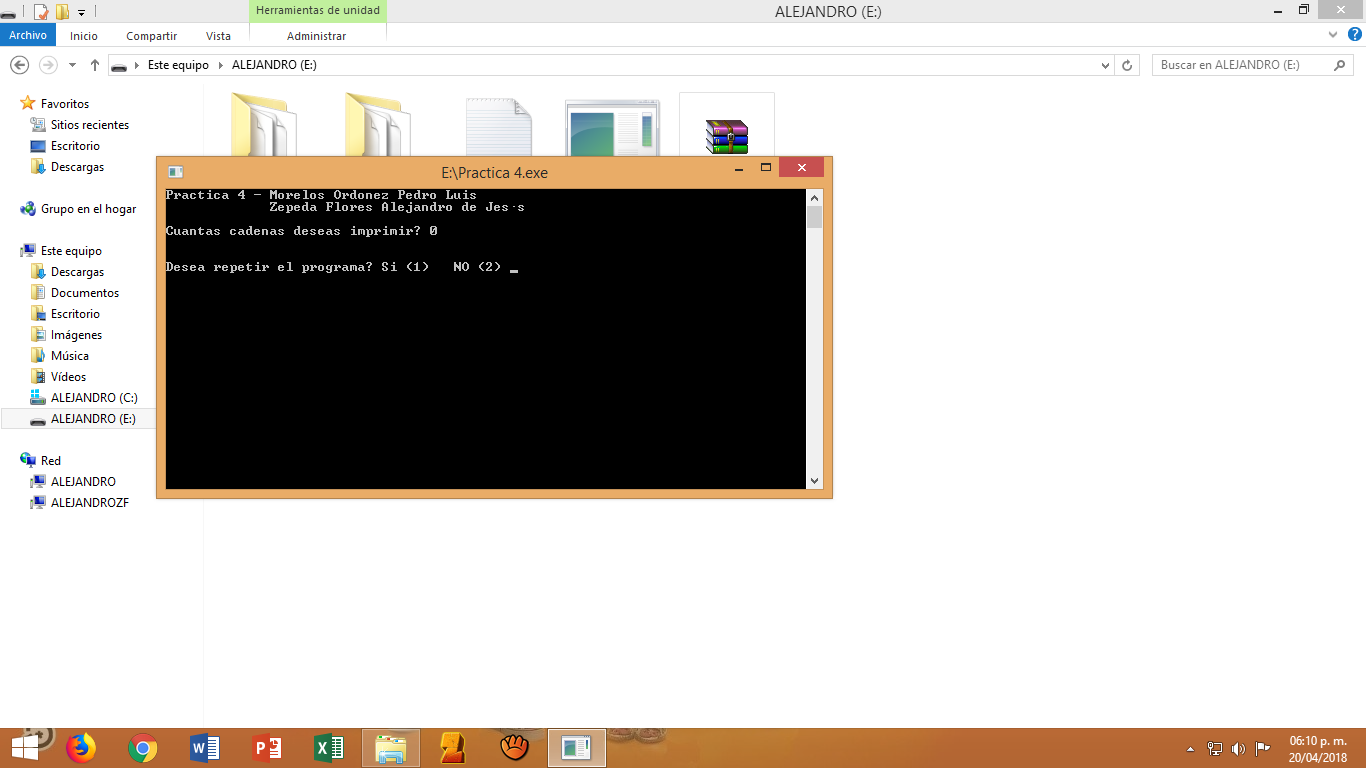
}

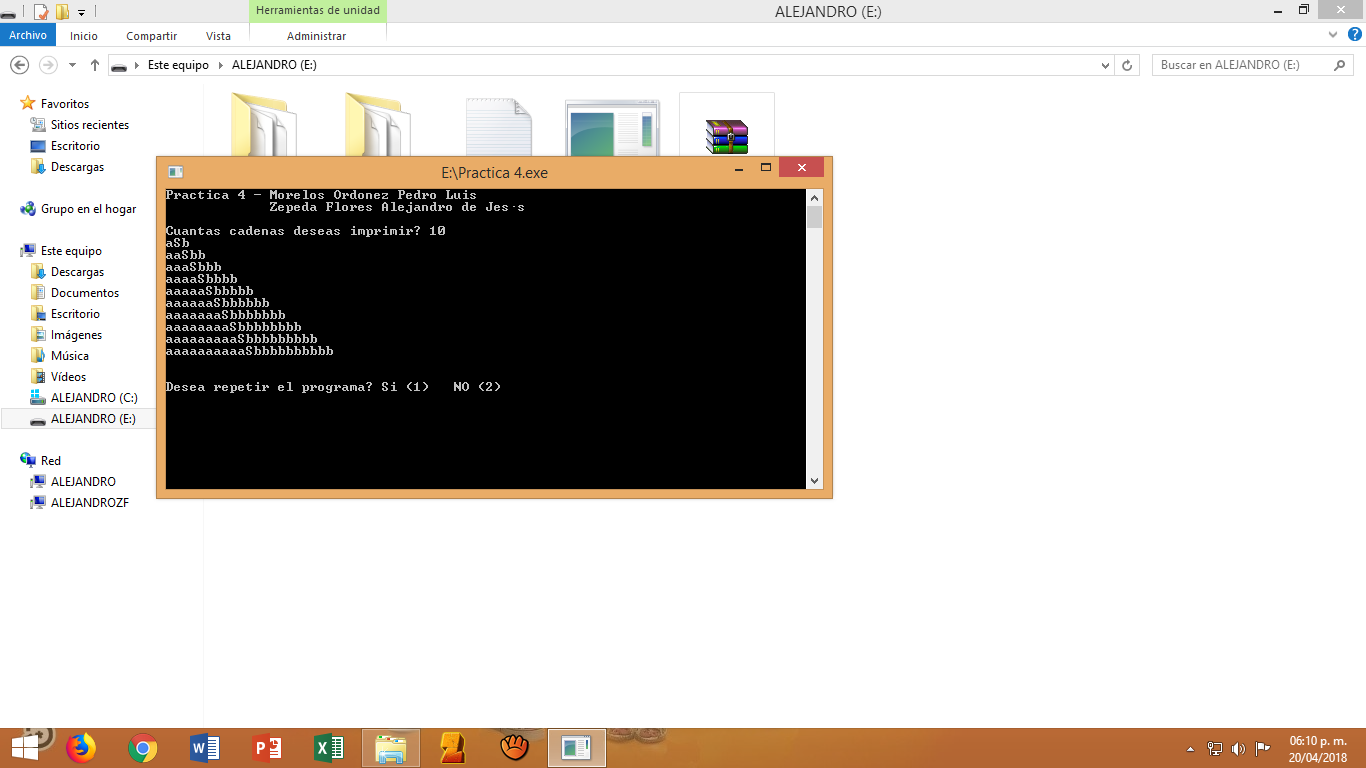
**FUNCIONAMIENTO**



Interfaz de usuario

****

****

****

**CONCLUSIÓN**

Fue una práctica interesante porque gracias a las gramáticas libres de contexto podemos generar mayor cantidad de lenguajes que con las gramáticas anteriores. Aunque presenta cierta complejidad generarlas porque tienen muchas reglas y estados que se deben cumplir; esto hace que se dificulte pero si se siguen las reglas al pie de la letra, generamos nuestro lenguaje.

**BIBLIOGRAFÍA**

* Ramón Brena. (2003). Autómatas y Lenguajes. Marzo 03, 2018, de Instituto Tecnológico y de Estudios Superiores de Monterrey Sitio web: http://fcbinueva.unillanos.edu.co/docus/Automatas%20Y%20LenguajeL.pdf
* Holger Billhardt. (2008). 1 Teoría de Autómatas y Lenguajes Formales. Marzo 02, 2018, de Universidad Rey Juan Carlos Sitio web: http://www.ia.urjc.es/grupo/docencia/automatas\_itis/apuntes/capitulo%201.ppt.pdf
* Franco Martínez Edgardo Adrián. (2013). Gramáticas libres de contexto. abril 20,2018, de Instituto Politécnico Nacional Sitio web: http://www.eafranco.com/docencia/teoriacomputacional/files/14/Clase\_14.pdf