REPORTE DE PRÁCTICAS SEGUNDO PARCIAL

ALUMNO: BASTIDA PRADO JAIME ARMANDO

PROFESOR: JUÁREZ MARTÍNEZ GENARO

GRUPO: 2CM5

Noviembre 2017

Índice

1. Menú Principal

1.1. Descripción

El menú principal es el programa desde el cual se podrá acceder a todos los demás programas del parcial 2, cuenta con una opción para abrir cada programa y la salida.

El menú principal tiene el siguiente aspecto:

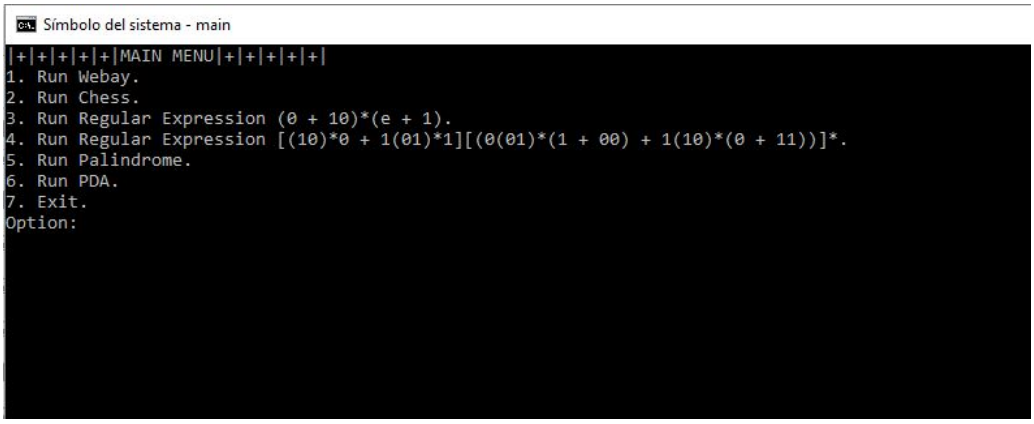


Figura 1: Menú Principal

1.2. Código

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

int main(void)
{
        int option = 0;

        system("cls");
        for(;;)
        {
                printf("|+|+|+|+|+|MAIN MENU|+|+|+|+|+|\n");
                printf("1. Run Webay.\n");
                printf("2. Run Chess.\n");
                printf("3. Run Regular Expression (0 + 10)*(e + 1)
                    .\n");
                printf("4. Run Regular Expression [(10)*0 + 1(01)
                    *1][(0(01)*(1 + 00) + 1(10)*(0 + 11))]*.\n");
                printf("5. Run Palindrome.\n");
                printf("6. Run PDA.\n");
                printf("7. Exit.\n");
                printf("Option: ");
                scanf(" %d", &option);
```

```
        switch(option)
        {
                case 1:
                        system("cls");
                        system("C:/Users/Jaime/Documents/
                            ESCOM_SEMESTRE_4/2
                            CM5_TEORIA_COMPUTACIONAL/UNIT_2
                            /WEBAY/a.exe");
                break;
                case 2:
                        system("cls");
                        system("C:/Users/Jaime/Documents/
                            ESCOM_SEMESTRE_4/2
                            CM5_TEORIA_COMPUTACIONAL/UNIT_2
                            /CHESS/chess.exe");
                break;
                case 3:
                        system("cls");
                        system("C:/Users/Jaime/Documents/
                            ESCOM_SEMESTRE_4/2
                            CM5_TEORIA_COMPUTACIONAL/UNIT_2
                            /RE_1/a.exe");
                break;
                case 4:
                        system("cls");
                        system("C:/Users/Jaime/Documents/
                            ESCOM_SEMESTRE_4/2
                            CM5_TEORIA_COMPUTACIONAL/UNIT_2
                            /RE_2/a.exe");
                break;
                case 5:
                        system("cls");
                        system("C:/Users/Jaime/Documents/
                            ESCOM_SEMESTRE_4/2
                            CM5_TEORIA_COMPUTACIONAL/UNIT_2
                            /CFG_1/a.exe");
                break;
                case 6:
                        system("cls");
                        system("C:/Users/Jaime/Documents/
                            ESCOM_SEMESTRE_4/2
                            CM5_TEORIA_COMPUTACIONAL/UNIT_2
                            /PDA/pda.exe");
                break;
                case 7:
                        system("cls");
                        exit(EXIT_SUCCESS);
                break;
                default:
                        system("cls");
                        printf("Choose a correct option.\n"
                            );
                        Sleep(2500);
                break;
        }
    }
    return 0;
```

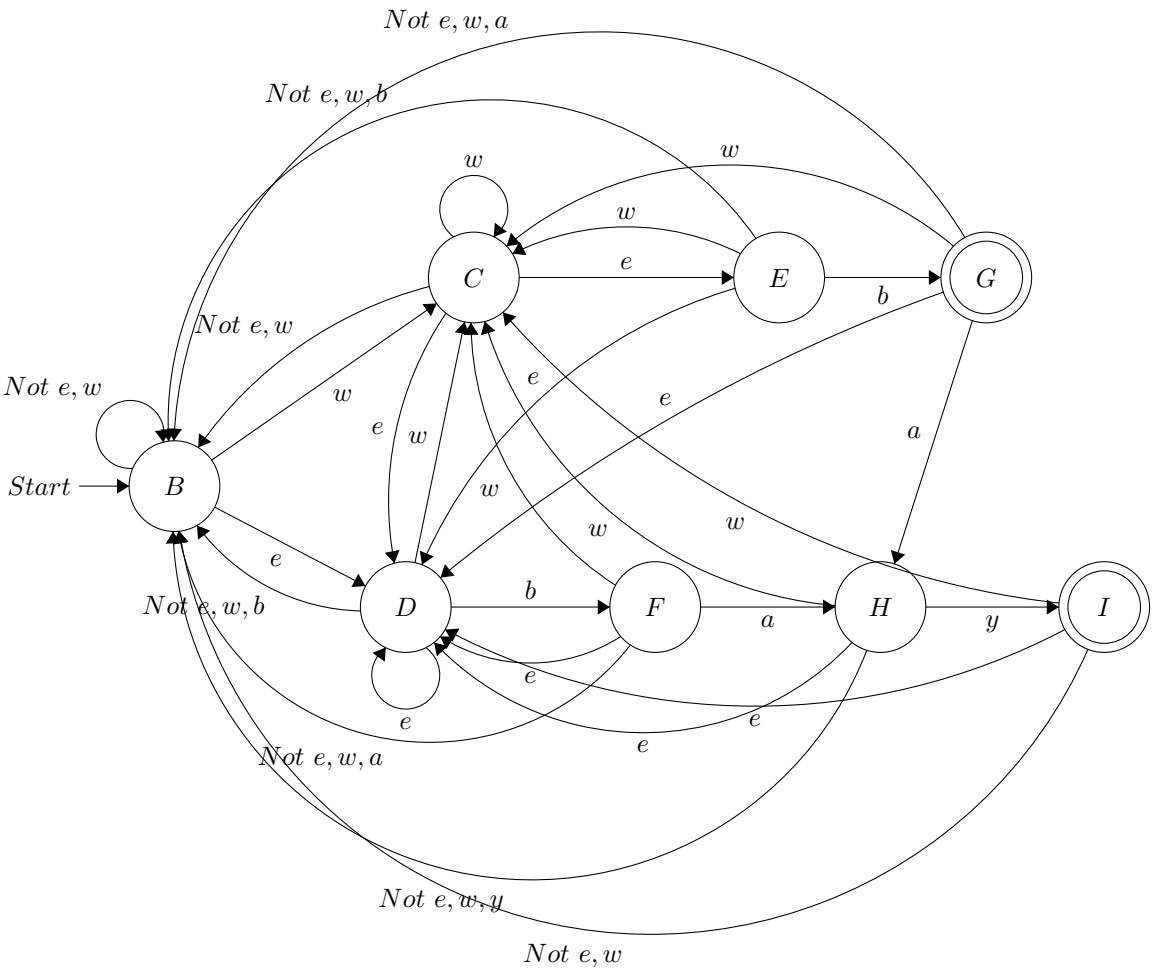
```
}
```

2. Práctica 1: Autómata finito determinístico que reconoce las palabras web y ebay.

2.1. Descripción

Este programa puede recibir y analizar un texto desde la consola o bien leer el texto desde un archivo proporcionando la ubicación en la cual se encuentra dicho archivo junto con su nombre. El programa mostrará todo el proceso que ejecutó el autómata y al final imprimirá cuantas y cuales fueron las palabras reconocidas, en caso que haya leído el texto desde un archivo también mostrará la posición en la cual se encontró cada palabra reconocida por el autómata, todo esto además de imprimirlo en consola lo enviará a un archivo que se podrá abrir desde el menú del programa para su revisión, por último el programa también tiene la capacidad de mostrar el grafo determinístico del autómata.

Grafo del automata:



2.2. Ejecución

Al iniciar el programa se nos presenta el menú con el siguiente aspecto:

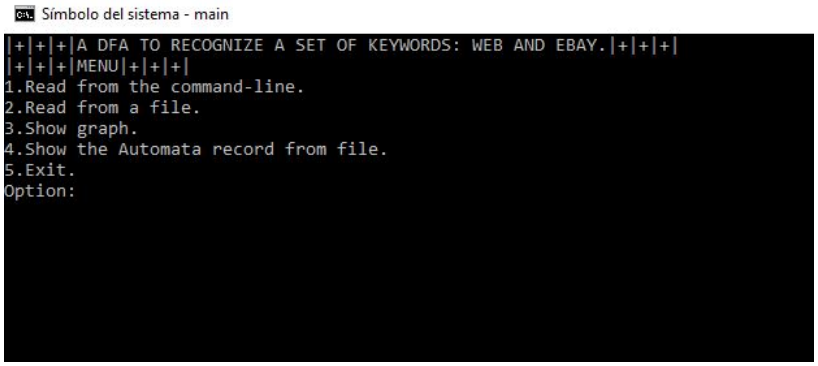


Figura 2: Menú Principal

Si elegimos la opción 1, el programa nos pedirá ingresar un texto y nos mostrará la salida correspondiente, además de mandar todo a un archivo:

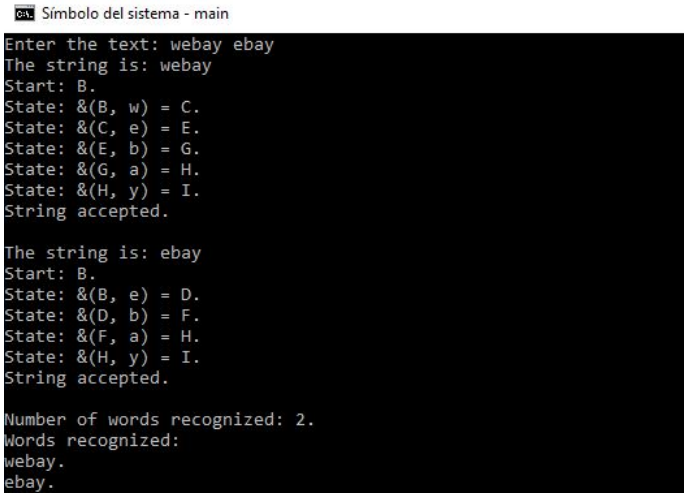


Figura 3: Opción 1

Si elegimos la opción 2, el programa nos pedirá ingresar la ubicación junto con el nombre de un archivo a leer y nos mostrará la salida correspondiente, además de mandar todo a un archivo:

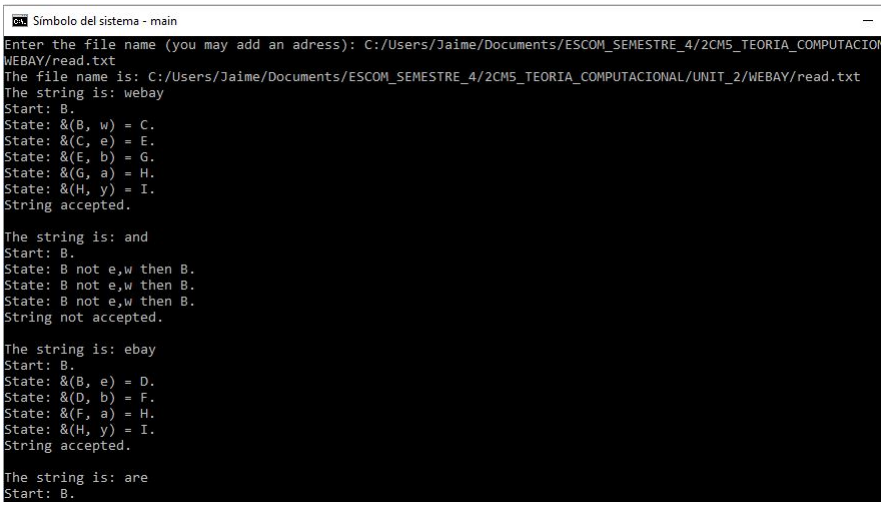


Figura 4: Opción 2

Si elegimos la opción 3, el programa mostrará el grafo del autómata:

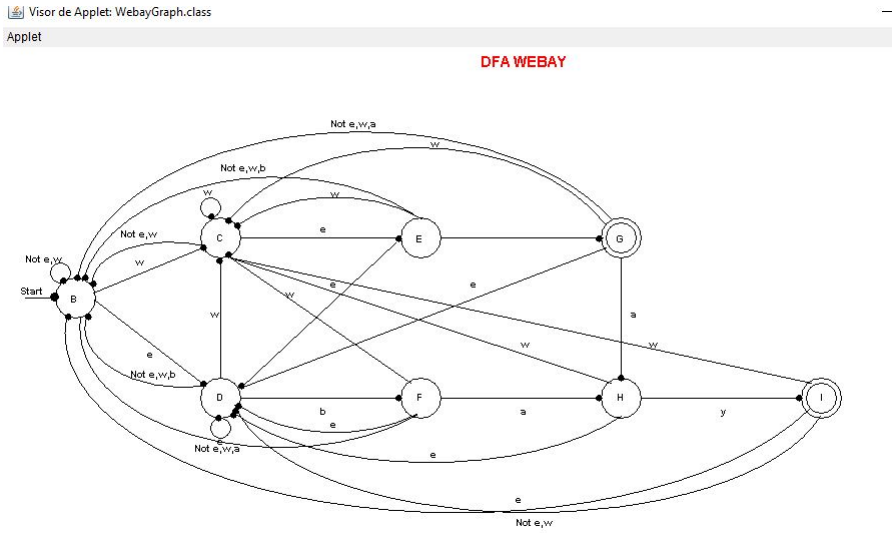


Figura 5: Opción 3

Si elegimos la opción 4, el programa mostrará el archivo con toda la información del proceso que realizó el autómata así como los datos relevantes mencionados en la descripción:

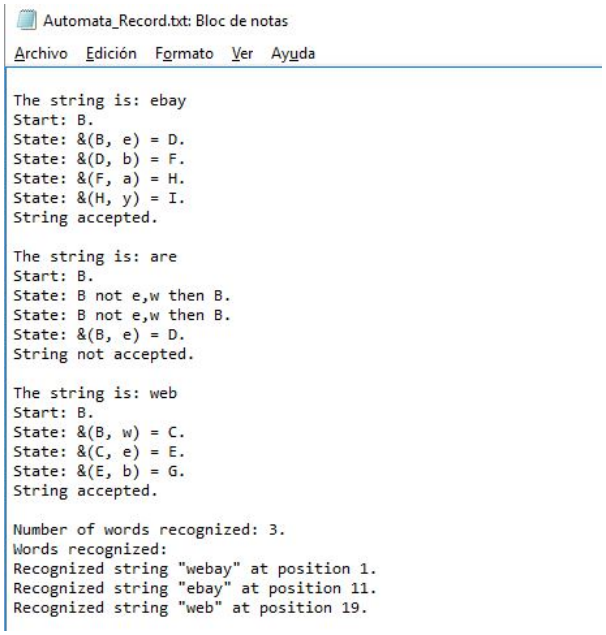


Figura 6: Opción 4

2.3. Código

```
#include <stdio.h>
#include <stdbool.h>
#include <ctype.h>
#include <stdlib.h>
#include <windows.h>
#include <string.h>

#define STR_LENGTH 100
#define MAX_RECO 100
#define B 0
#define C 1
#define D 2
#define E 3
#define F 4
#define G 5
#define H 6
#define I 7

int getString(char *string, int n);
int getStringFile(char *string, int n, FILE *read_fp);
bool automata(char *ch, FILE *write_fp);

int main (void)
{
        int i = 0, j = 0, ch, last_ch, counter = 0, option = 0;
        char string[STR_LENGTH + 1] = {'\0'};
        char *recognized_strings[MAX_RECO];
        long recognized_positions[MAX_RECO] = {0};
        char file_name[FILENAME_MAX]; //DEFINED IN <stdio.h>
        FILE *read_fp, *write_fp;

        system("cls");
        for(;;)
        {
                printf("|+|+|+|A DFA TO RECOGNIZE A SET OF KEYWORDS
                    : WEB AND EBAY.|+|+|+|\n");
                printf("|+|+|+|MENU|+|+|+|\n");
                printf("1.Read from the command-line.\n");
                printf("2.Read from a file.\n");
                printf("3.Show graph.\n");
                printf("4.Show the Automata record from file.\n");
                printf("5.Exit.\n");
                printf("Option: ");
                scanf(" %d", &option);
                system("cls");
                switch(option)
                {
                        case 1:
                        counter = 0;
                        fflush(stdin); //CLEANS THE BUFFER TO
                            PREVENT READING '\n' IN THE FIRST TRY
                        printf("Enter the text: ");
                        if((write_fp = fopen("C:/Users/Jaime/
                            Documents/ESCOM_SEMESTRE_4/2
                            CM5_TEORIA_COMPUTACIONAL/UNIT_2/WEBAY/
```

```
    Automata_Record.txt", "w")) == NULL)
{
fprintf(stderr, "Can't write to file.\n");
break;
}
for(;;)
{
last_ch = getString(string, STR_LENGTH);
if(string[0] != '\0')
{
printf("The string is: %s\n", string);
fprintf(write_fp, "The string is: %s\n",
    string);
if(automata(string, write_fp))
{
recognized_strings[counter] = malloc(strlen
    (string) + 1); //WE ADD "+ 1" BECAUSE
    OF THE '\0' AT THE END OF EACH STRING
strcpy(recognized_strings[counter], string)
    ;
counter += 1;
}
}
if(last_ch == '\n')
        break;
}
printf("Number of words recognized: %d.\n",
     counter);
fprintf(write_fp, "Number of words
    recognized: %d.\n", counter);
printf("Words recognized: \n");
fprintf(write_fp, "Words recognized: \n");
for(i = 0; i < counter; i++)
{
printf("%s.\n", recognized_strings[i]);
fprintf(write_fp, "%s.\n",
    recognized_strings[i]);
}
fclose(write_fp);
break;
case 2:
counter = 0;
fflush(stdin); //CLEANS THE BUFFER TO
    PREVENT READING '\n' IN THE FIRST TRY
printf("Enter the file name (you may add an
    adress): ");
scanf("%s", file_name);
printf("The file name is: %s\n", file_name)
    ;
if((read_fp = fopen(file_name, "rb")) ==
    NULL)
{
fprintf(stderr, "Can't open: %s\n",
    file_name);
break;
}
```

```
if((write_fp = fopen("C:/Users/Jaime/
    Documents/ESCOM_SEMESTRE_4/2
    CM5_TEORIA_COMPUTACIONAL/UNIT_2/WEBAY/
    Automata_Record.txt", "w")) == NULL)
{
fprintf(stderr, "Can't write to file.\n");
break;
}
for(;;)
{
last_ch = getStringFile(string, STR_LENGTH,
     read_fp);
if(string[0] != '\0')
{
printf("The string is: %s\n", string);
fprintf(write_fp, "The string is: %s\n",
    string);
if(automata(string, write_fp))
{
recognized_strings[counter] = malloc(strlen
    (string) + 1);
recognized_positions[counter] = (ftell(
    read_fp) - strlen(string));
strcpy(recognized_strings[counter], string)
    ;
counter += 1;
}
}
if(last_ch == EOF)
break;
}
fclose(read_fp);
printf("Number of words recognized: %d.\n",
     counter);
fprintf(write_fp, "Number of words
    recognized: %d.\n", counter);
printf("Words recognized: \n");
fprintf(write_fp, "Words recognized: \n");
for(i = 0; i < counter; i++)
{
printf("Recognized string \"%s\" at
    position %d.\n", recognized_strings[i
    ], recognized_positions[i]);
fprintf(write_fp, "Recognized string \"%s\"
    at position %d.\n",
    recognized_strings[i],
    recognized_positions[i]);
}
fclose(write_fp);
break;
case 3:
system("appletviewer WebayGraph.html");
break;
case 4:
system("C:/Users/Jaime/Documents/
    ESCOM_SEMESTRE_4/2
    CM5_TEORIA_COMPUTACIONAL/UNIT_2/WEBAY/
```

```
                                Automata_Record.txt");
                        break;
                        case 5:
                        system("cls");
                        exit(EXIT_SUCCESS);
                        break;
                        default:
                        printf("Choose a correct option.\n");
                        Sleep(2000);
                        break;
                }
        }
        return 0;
}

int getString(char *string, int max_length)
{
        int i = 0, ch;

        while (((ch = getchar()) != ' ') && ((ch >= 65 && ch <= 90)
             || (ch >= 97 && ch <= 122)) && i < max_length && ch !=
             '\n')
                string[i++] = ch;
        string[i] = '\0';

        return ch;
}

int getStringFile(char *string, int max_length, FILE *read_fp)
{
        int i = 0, ch;

        while (((ch = getc(read_fp)) != ' ') && ((ch >= 65 && ch <=
             90) || (ch >= 97 && ch <= 122)) && i < max_length &&
            ch != EOF)
                string[i++] = ch;
        string[i] = '\0';

        return ch;
}

bool automata(char *ch, FILE *write_fp)
{
        int i = 0, state = B;

        printf("Start: B.\n");
        fprintf(write_fp, "Start: B.\n");
        //WE HAVE GUARANTEED THAT NON EMPTY STRINGS WILL STEP
            THROUGH THIS POINT SO WE CAN USE DO-WHILE
        do
        {
                //HANDLES ALL POSIBILITIES WHILE BEING IN STATE B
                if(state == B && toupper(ch[i]) == 'W')
                {
                        printf("State: &(B, w) = C.\n");
                        fprintf(write_fp, "State: &(B, w) = C.\n");
                        state = C;
```

```
}
else if(state == B && toupper(ch[i]) == 'E')
{
        printf("State: &(B, e) = D.\n");
        fprintf(write_fp, "State: &(B, e) = D.\n");
        state = D;
}
else if (state == B)
{
        printf("State: B not e, w then B.\n");
        fprintf(write_fp, "State: B not e, w then B
            .\n");
        state = B;
}
//HANDLES ALL POSIBILITIES WHILE BEING IN STATE C
else if(state == C && toupper(ch[i]) == 'E')
{
        printf("State: &(C, e) = E.\n");
        fprintf(write_fp, "State: &(C, e) = E.\n");
        state = E;
}
else if(state == C && toupper(ch[i]) == 'W')
{
        printf("State: &(C, w) = C.\n");
        fprintf(write_fp, "State: &(C, w) = C.\n");
        state = C;
}
else if (state == C)
{
        printf("State: C not e, w then B.\n");
        fprintf(write_fp, "State: C not e, w then B
            .\n");
        state = B;
}
//HANDLES ALL POSIBILITIES WHILE BEING IN STATE D
else if(state == D && toupper(ch[i]) == 'B')
{
        printf("State: &(D, b) = F.\n");
        fprintf(write_fp, "State: &(D, b) = F.\n");
        state = F;
}
else if(state == D && toupper(ch[i]) == 'E')
{
        printf("State: &(D, e) = D.\n");
        fprintf(write_fp, "State: &(D, e) = D.\n");
        state = D;
}
else if(state == D && toupper(ch[i]) == 'W')
{
        printf("State: &(D, w) = C.\n");
        fprintf(write_fp, "State: &(D, w) = C.\n");
        state = C;
}
else if (state == D)
{
        printf("State: D not e, w, b then B.\n");
```

```
        fprintf(write_fp, "State: D not e,w,b then
            B.\n");
        state = B;
}
//HANDLES ALL POSIBILITIES WHILE BEING IN STATE E
else if(state == E && toupper(ch[i]) == 'B')
{
        printf("State: &(E, b) = G.\n");
        fprintf(write_fp, "State: &(E, b) = G.\n");
        state = G;
}
else if(state == E && toupper(ch[i]) == 'W')
{
        printf("State: &(E, w) = C.\n");
        fprintf(write_fp, "State: &(E, w) = C.\n");
        state = C;
}
else if(state == E && toupper(ch[i]) == 'E')
{
        printf("State: &(E, e) = D.\n");
        fprintf(write_fp, "State: &(E, e) = D.\n");
        state = D;
}
else if (state == E)
{
        printf("State: D not e,w,b then B.\n");
        fprintf(write_fp, "State: D not e,w,b then
            B.\n");
        state = B;
}
//HANDLES ALL POSIBILITIES WHILE BEING IN STATE F
else if(state == F && toupper(ch[i]) == 'A')
{
        printf("State: &(F, a) = H.\n");
        fprintf(write_fp, "State: &(F, a) = H.\n");
        state = H;
}
else if(state == F && toupper(ch[i]) == 'E')
{
        printf("State: &(F, e) = D.\n");
        fprintf(write_fp, "State: &(F, e) = D.\n");
        state = D;
}
else if(state == F && toupper(ch[i]) == 'W')
{
        printf("State: &(F, w) = C.\n");
        fprintf(write_fp, "State: &(F, w) = C.\n");
        state = C;
}
else if (state == F)
{
        printf("State: F not e,w,a then B.\n");
        fprintf(write_fp, "State: F not e,w,a then
            B.\n");
        state = B;
}
//HANDLES ALL POSIBILITIES WHILE BEING IN STATE G
```

```
else if(state == G && toupper(ch[i]) == 'A')
{
        printf("State: &(G, a) = H.\n");
        fprintf(write_fp, "State: &(G, a) = H.\n");
        state = H;
}
else if(state == G && toupper(ch[i]) == 'E')
{
        printf("State: &(G, e) = D.\n");
        fprintf(write_fp, "State: &(G, e) = D.\n");
        state = D;
}
else if(state == G && toupper(ch[i]) == 'W')
{
        printf("State: &(G, w) = C.\n");
        fprintf(write_fp, "State: &(G, w) = C.\n");
        state = C;
}
else if (state == G)
{
        printf("State: G not e,w,a then B.\n");
        fprintf(write_fp, "State: G not e,w,a then
            B.\n");
        state = B;
}
//HANDLES ALL POSIBILITIES WHILE BEING IN STATE H
else if(state == H && toupper(ch[i]) == 'Y')
{
        printf("State: &(H, y) = I.\n");
        fprintf(write_fp, "State: &(H, y) = I.\n");
        state = I;
}
else if(state == H && toupper(ch[i]) == 'E')
{
        printf("State: &(H, e) = D.\n");
        fprintf(write_fp, "State: &(H, e) = D.\n");
        state = D;
}
else if(state == H && toupper(ch[i]) == 'W')
{
        printf("State: &(H, w) = C.\n");
        fprintf(write_fp, "State: &(H, w) = C.\n");
        state = C;
}
else if (state == H)
{
        printf("State: H not e,w,y then B.\n");
        fprintf(write_fp, "State: H not e,w,y then
            B.\n");
        state = B;
}
//HANDLES ALL POSIBILITIES WHILE BEING IN STATE I
else if(state == I && toupper(ch[i]) == 'W')
{
        printf("State: &(I, w) = C.\n");
        fprintf(write_fp, "State: &(I, w) = C.\n");
        state = C;
```

```
                }
                else if(state == I && toupper(ch[i]) == 'E')
                {
                        printf("State: &(I, e) = D.\n");
                        fprintf(write_fp, "State: &(I, e) = D.\n");
                        state = D;
                }
                else if (state == I)
                {
                        printf("State: I not e, w then B.\n");
                        fprintf(write_fp, "State: I not e, w then B
                            .\n");
                        state = B;
                }
                //HANDLES FINAL STATES
                if(state == G && ch[i + 1] == '\0')
                {
                        printf("String accepted.\n\n");
                        fprintf(write_fp, "String accepted.\n\n");
                        return true;
                }
                if(state == I && ch[i + 1] == '\0')
                {
                        printf("String accepted.\n\n");
                        fprintf(write_fp, "String accepted.\n\n");
                        return true;
                }
                i++;
        }
        while(ch[i] != '\0');

        printf("String not accepted.\n\n");
        fprintf(write_fp, "String not accepted.\n\n");
        return false;
}
```

3. Práctica 2: Autómata finito no determinístico que se mueve en un tablero de ajedrez de 3x3.

3.1. Descripción

Este programa recibe una cadena desde consola, desde archivo (proporcionando la ubicación junto con el nombre), o generando una cadena aleatoria de r's y b's con un tope de 1000 carácteres (el usuario puede ingresar la longitud de esta cadena). El tablero se divide en casillas rojas y negras con 2,4,6 y 8 rojas, las demás negras. El estado incial del autómata es la casilla 1 y el final la casilla 9, por cada r que lee el autómata este parte de la casilla en la que se encuentra a todas las casillas adyacentes rojas, y lo correpondiente con b(casillas negras), dividiéndose así en varios caminos a medida que lee la cadena. Si la cadena se ha consumido y alguno de todos los caminos que tomó el autómata terminó en la casilla 9 se dice que la cadena fue exitosa, existiendo diversos caminos exitosos y fracasos.

El programa nos dirá si la cadena fue exitosa, mandando a un archivo llamado: SuccesfulTrails.txt solo 3 de todos los caminos exitosos (en caso de existir) y a otro archivo llamado: Trails.txt todos los demás caminos.

Por último señalar que el programa implementa el tipo de dato abstracto Arbol, cuyo código se encuentra al final de esta sección.

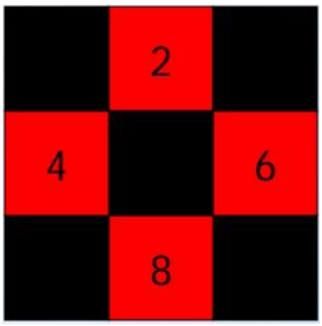
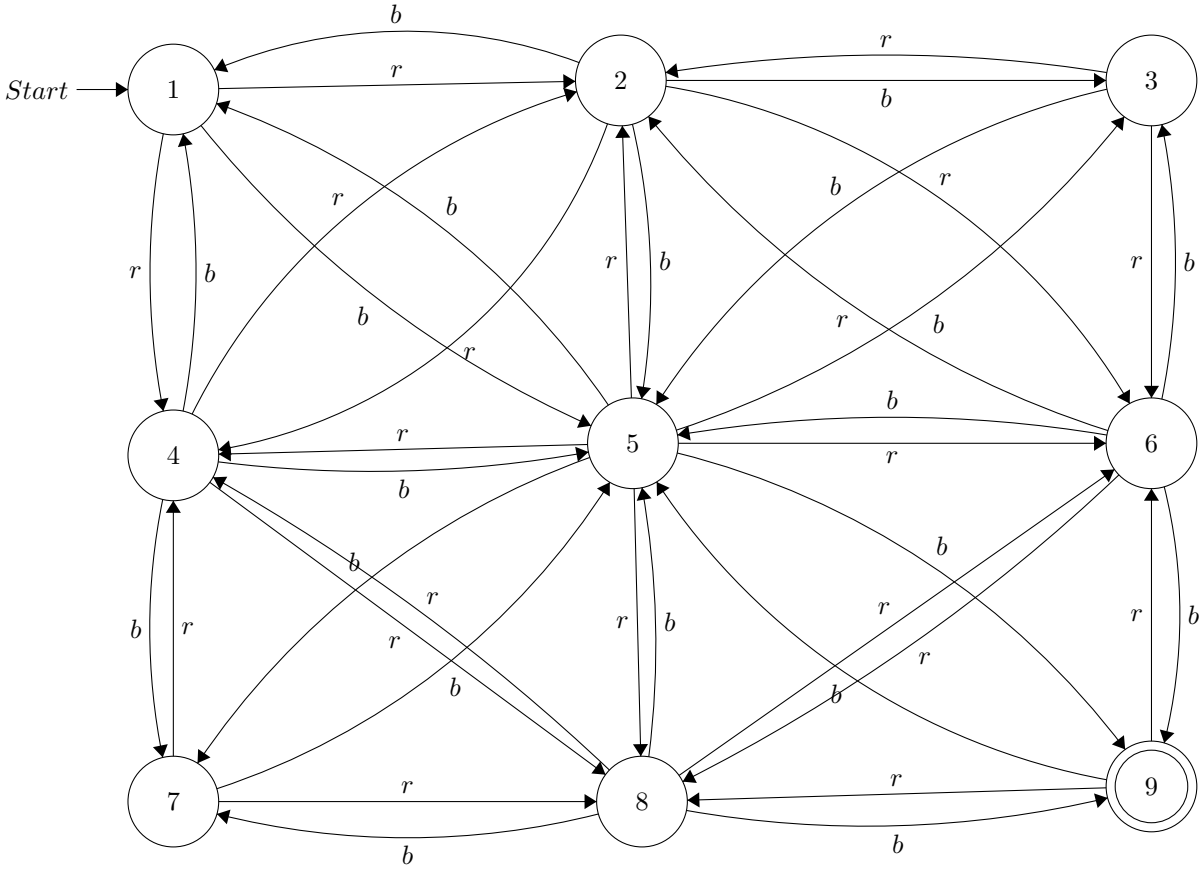


Figura 7: Tablero

Grafo del automata:



3.2. Ejecución

Al iniciar el programa se nos presenta el menú con el siguiente aspecto:

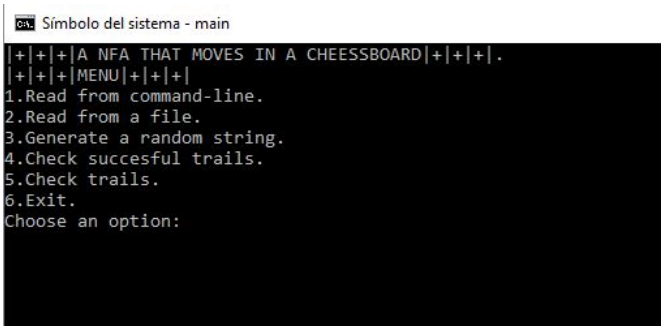


Figura 8: Menú Principal

Si elegimos la opción 1, el programa nos pedirá ingresar una cadena y nos mostrará la salida correspondiente, además de mandar todo a los archivos correspondientes:

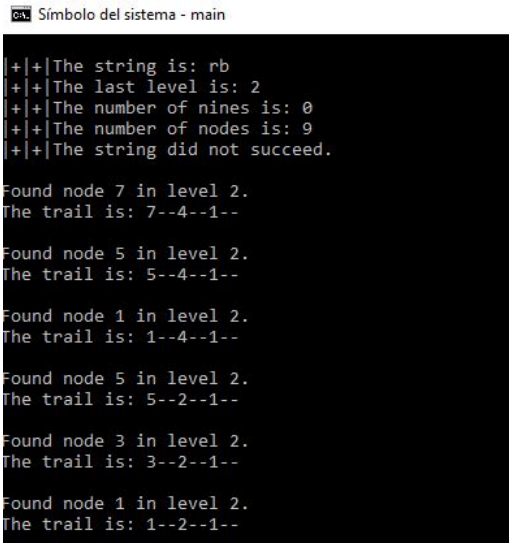


Figura 9: Opción 1

Si elegimos la opción 2, el programa nos pedirá ingresar la ubicación junto con el nombre de un archivo a leer y nos mostrará la salida correspondiente, además de mandar todo a los archivos correspondientes:

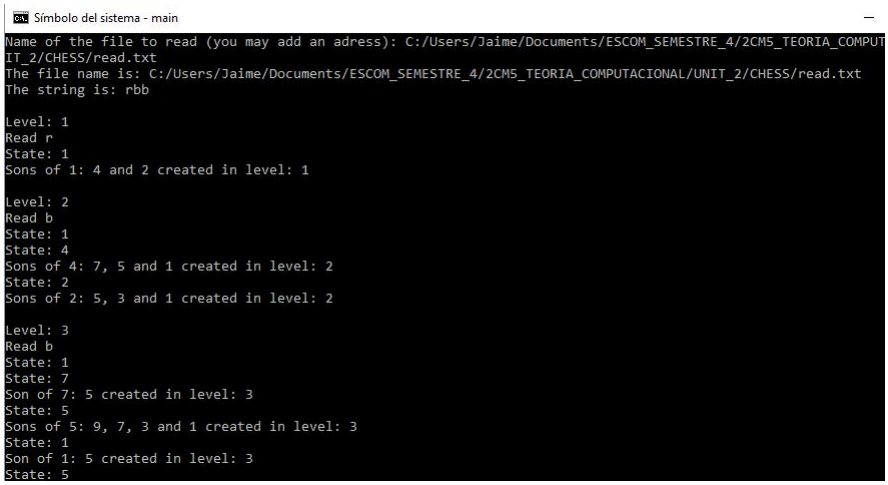


Figura 10: Opción 2

Si elegimos la opción 3, el programa nos pedirá ingresar una longitud para la cadena aleatoria, la generará y la procesará, haciendo lo mismo que en las opciones anteriores:

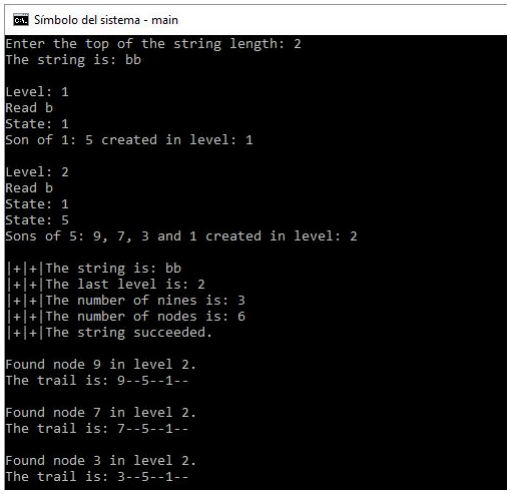


Figura 11: Opción 3

Si elegimos la opción 4, el programa mostrará el archivo con los caminos exitosos:

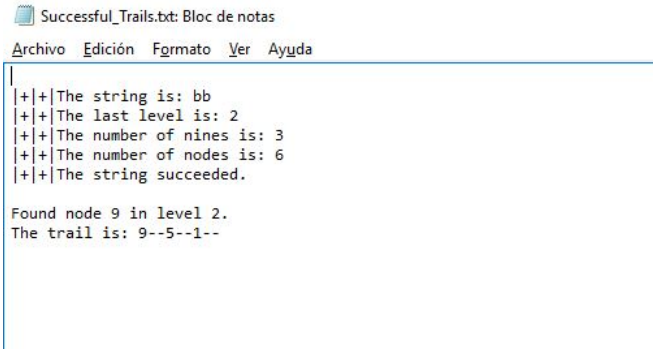


Figura 12: Opción 4

Si elegimos la opción 5, el programa mostrará el archivo con los todos los caminos:

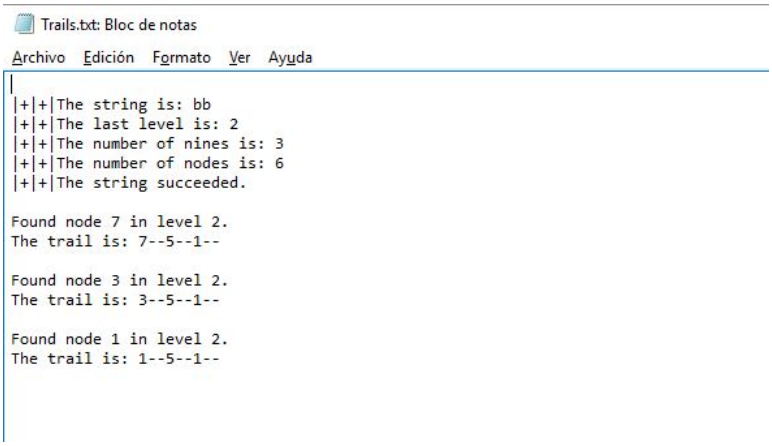


Figura 13: Opción 5

3.3. Código

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <time.h>
#include "TADArbol.h"

#define TRACE_LENGTH 1000

int main (void)
{
        int i = 0, levels = 0, inf_level = 0, j = 0, nines = 0,
            option = 0, nodes = 0, successfuls = 3, top_random = 0,
             random_ch;
        char string[100] = {'\0'};
        Tree tree;
        Position position, father;
        element state;
        bool first_time = true, proceed = true;
        int trace[TRACE_LENGTH];
        FILE *read_fp, *write_trails_fp, *write_success_fp;
        char file_name[FILENAME_MAX];
        srand((unsigned) time(NULL));

        for(;;)
        {
                printf("|+|+|+|A NFA THAT MOVES IN A CHEESSBOARD
                    |+|+|+|.\n");
                printf("|+|+|+|MENU|+|+|+|\n");
                printf("1.Read from command-line.\n");
                printf("2.Read from a file.\n");
                printf("3.Generate a random string.\n");
                printf("4.Check succesful trails.\n");
                printf("5.Check trails.\n");
                printf("6.Exit.\n");
                printf("Choose an option: ");
                scanf(" %d", &option);

                if(option == 1)
                {
                        system("cls");
                        printf("Enter the string: ");
                        scanf("%s", string);
                        printf("The string is: %s\n", string);
                        proceed = true;
                }
                else if(option == 2)
                {
                        system("cls");
                        printf("Name of the file to read (you may
                            add an adress): ");
                        scanf("%s", file_name);
                        printf("The file name is: %s\n");
                        if((read_fp = fopen(file_name, "rb")) ==
                            NULL)
                        {
```

```
            printf("Can't read %s.\n", file_name);
            }
            fscanf(read_fp, "%s", string);
            printf("The string is: %s\n", string);
            proceed = true;
            fclose(read_fp);
    }
    else if(option == 3)
    {
            system("cls");
            printf("Enter the top of the string length:
                ");
            scanf("%d", &top_random);
            for(i = 0; i < top_random; i++)
            {
            random_ch = rand() % 2;
            if(random_ch == 0)
            string[i] = 'r';
            else if(random_ch == 1)
            string[i] = 'b';
            }
            string[i] = '\0';
            printf("The string is: %s\n", string);
            proceed = true;
    }
    else if(option == 4)
    {
            system("C:/Users/Jaime/Documents/
                ESCOM_SEMESTRE_4/2
                CM5_TEORIA_COMPUTACIONAL/UNIT_2/CHESS/
                Successful_Trails.txt");
            proceed = false;
    }
    else if(option == 5)
    {
            system("C:/Users/Jaime/Documents/
                ESCOM_SEMESTRE_4/2
                CM5_TEORIA_COMPUTACIONAL/UNIT_2/CHESS/
                Trails.txt");
            proceed = false;
    }
    else if(option == 6)
    {
            system("cls");
            exit(EXIT_SUCCESS);
    }
    else
    {
            system("cls");
            printf("Choose a correct option.\n");
            Sleep(2500);
            proceed = false;
    }

    if(proceed == true)
    {
            levels = 0;
```

```
nodes = 0;
successfuls = 3;
Initialize(&tree);
NewRightSon(&tree, position, (element) {.c
    = '1', .visited = false, .level =
    levels});
nodes++;
position = Root(&tree);
i = 0;
while(string[i] != '\0')
{
levels++;
printf("\nLevel: %d\n", levels);
if(string[i] == 'r')
{
printf("Read r\n");
for(position = Root(&tree); position !=
    NULL; position = SearchNonVisited(&tree
    , inf_level)) //Visits all nodes
{
printf("State: %c\n", position->e.c);
if(position->e.c == '1' && position->e.
    visited == false) //Just this one has
    the second condition because if the
    first symbol in the chain is r
{
position->e.visited = true;
NewRightSon(&tree, position, (element) {.c
    = '4', .visited = false, .level =
    levels});
NewMiddleRightSon(&tree, position, (element
    ) {.c = '2', .visited = false, .level =
     levels});
printf("Sons of 1: 4 and 2 created in level
    : %d\n", levels);
nodes += 2;
}
else if(position->e.c == '2')
{
position->e.visited = true;
NewRightSon(&tree, position, (element) {.c
    = '6', .visited = false, .level =
    levels});
NewMiddleRightSon(&tree, position, (element
    ) {.c = '4', .visited = false, .level =
     levels});
printf("Sons of 2: 6 and 4 created in level
    : %d\n", levels);
nodes += 2;
}
else if(position->e.c == '3')
{
position->e.visited = true;
NewRightSon(&tree, position, (element) {.c
    = '6', .visited = false, .level =
    levels});
```

```
NewMiddleRightSon(&tree, position, (element
    ) {.c = '2', .visited = false, .level =
     levels});
printf("Sons of 3: 6 and 2 created in level
    : %d\n", levels);
nodes += 2;
}
else if(position->e.c == '4')
{
position->e.visited = true;
NewRightSon(&tree, position, (element) {.c
    = '8', .visited = false, .level =
    levels});
NewMiddleRightSon(&tree, position, (element
    ) {.c = '2', .visited = false, .level =
     levels});
printf("Sons of 4: 8 and 2 created in level
    : %d\n", levels);
nodes += 2;
}
else if(position->e.c == '5')
{
position->e.visited = true;
NewRightSon(&tree, position, (element) {.c
    = '8', .visited = false, .level =
    levels});
NewMiddleRightSon(&tree, position, (element
    ) {.c = '6', .visited = false, .level =
     levels});
NewMiddleLeftSon(&tree, position, (element)
     {.c = '4', .visited = false, .level =
    levels});
NewLeftSon(&tree, position, (element) {.c =
     '2', .visited = false, .level = levels
    });
printf("Sons of 5: 8, 6, 4 and 2 created in
    level: %d\n", levels);
nodes += 4;
}
else if(position->e.c == '6')
{
position->e.visited = true;
NewRightSon(&tree, position, (element) {.c
    = '8', .visited = false, .level =
    levels});
NewMiddleRightSon(&tree, position, (element
    ) {.c = '2', .visited = false, .level =
     levels});
printf("Sons of 6: 8 and 2 created in level
    : %d\n", levels);
nodes += 2;
}
else if(position->e.c == '7')
{
position->e.visited = true;
NewRightSon(&tree, position, (element) {.c
    = '8', .visited = false, .level =
```

```
    levels});
NewMiddleRightSon(&tree, position, (element
    ) {.c = '4', .visited = false, .level =
     levels});
printf("Sons of 7: 8 and 4 created in level
    : %d\n", levels);
nodes += 2;
}
else if(position->e.c == '8')
{
position->e.visited = true;
NewRightSon(&tree, position, (element) {.c
    = '6', .visited = false, .level =
    levels});
NewMiddleRightSon(&tree, position, (element
    ) {.c = '4', .visited = false, .level =
     levels});
printf("Sons of 8: 6 and 4 created in level
    : %d\n", levels);
nodes += 2;
}
else if(position->e.c == '9')
{
position->e.visited = true;
NewRightSon(&tree, position, (element) {.c
    = '8', .visited = false, .level =
    levels});
NewMiddleRightSon(&tree, position, (element
    ) {.c = '6', .visited = false, .level =
     levels});
printf("Sons of 9: 8 and 6 created in level
    : %d\n", levels);
nodes += 2;
}
if(first_time)
{
first_time = false;
break;
}
inf_level = levels - 1;
}
}
else if(string[i] == 'b')
{
printf("Read b\n");
for(position = Root(&tree); position !=
    NULL; position = SearchNonVisited(&tree
    , inf_level)) //Visits all nodes
{
printf("State: %c\n", position->e.c);
if(position->e.c == '1' && position->e.
    visited == false)
{
position->e.visited = true;
NewRightSon(&tree, position, (element) {.c
    = '5', .visited = false, .level =
    levels});
```

```
printf("Son␣of␣1:␣5␣created␣in␣level:␣%d\n"
    , levels);
nodes++;
}
else if(position->e.c == '2')
{
position->e.visited = true;
NewRightSon(&tree, position, (element) {.c
    = '5', .visited = false, .level =
    levels});
NewMiddleRightSon(&tree, position, (element
    ) {.c = '3', .visited = false, .level =
     levels});
NewMiddleLeftSon(&tree, position, (element)
     {.c = '1', .visited = false, .level =
    levels});
printf("Sons␣of␣2:␣5,␣3␣and␣1␣created␣in␣
    level:␣%d\n", levels);
nodes += 3;
}
else if(position->e.c == '3')
{
position->e.visited = true;
NewRightSon(&tree, position, (element) {.c
    = '5', .visited = false, .level =
    levels});
printf("Son␣of␣3:␣5␣created␣in␣level:␣%d\n"
    , levels);
nodes++;
}
else if(position->e.c == '4')
{
position->e.visited = true;
NewRightSon(&tree, position, (element) {.c
    = '7', .visited = false, .level =
    levels});
NewMiddleRightSon(&tree, position, (element
    ) {.c = '5', .visited = false, .level =
     levels});
NewMiddleLeftSon(&tree, position, (element)
     {.c = '1', .visited = false, .level =
    levels});
printf("Sons␣of␣4:␣7,␣5␣and␣1␣created␣in␣
    level:␣%d\n", levels);
nodes += 3;
}
else if(position->e.c == '5')
{
position->e.visited = true;
NewRightSon(&tree, position, (element) {.c
    = '9', .visited = false, .level =
    levels});
NewMiddleRightSon(&tree, position, (element
    ) {.c = '7', .visited = false, .level =
     levels});
NewMiddleLeftSon(&tree, position, (element)
     {.c = '3', .visited = false, .level =
```

```
    levels});
NewLeftSon(&tree, position, (element) {.c =
     '1', .visited = false, .level = levels
    });
printf("Sons of 5: 9, 7, 3 and 1 created in
    level: %d\n", levels);
nines++;
nodes += 4;
}
else if(position->e.c == '6')
{
position->e.visited = true;
NewRightSon(&tree, position, (element) {.c
    = '9', .visited = false, .level =
    levels});
NewMiddleRightSon(&tree, position, (element
    ) {.c = '5', .visited = false, .level =
     levels});
NewMiddleLeftSon(&tree, position, (element)
     {.c = '3', .visited = false, .level =
    levels});
printf("Son of 6: 9, 5 and 3 created in
    level: %d\n", levels);
nines++;
nodes += 3;
}
else if(position->e.c == '7')
{
position->e.visited = true;
NewRightSon(&tree, position, (element) {.c
    = '5', .visited = false, .level =
    levels});
printf("Son of 7: 5 created in level: %d\n"
    , levels);
nodes++;
}
else if(position->e.c == '8')
{
position->e.visited = true;
NewRightSon(&tree, position, (element) {.c
    = '9', .visited = false, .level =
    levels});
NewMiddleRightSon(&tree, position, (element
    ) {.c = '7', .visited = false, .level =
     levels});
NewMiddleLeftSon(&tree, position, (element)
     {.c = '5', .visited = false, .level =
    levels});
printf("Son of 8: 9, 7 and 5 created in
    level: %d\n", levels);
nines++;
nodes += 3;
}
else if(position->e.c == '9')
{
position->e.visited = true;
```

```
NewRightSon(&tree, position, (element) {.c
    = '5', .visited = false, .level =
    levels});
printf("Son␣of␣9:␣5␣created␣in␣level:␣%d\n"
    , levels);
nodes++;
}
if(first_time)
{
first_time = false;
break;
}
inf_level = levels - 1;
}
}
i++;
}

if((write_trails_fp = fopen("C:/Users/Jaime
    /Documents/ESCOM_SEMESTRE_4/2
    CM5_TEORIA_COMPUTACIONAL/UNIT_2/CHESS/
    Trails.txt", "w")) == NULL)
{
printf("Can't␣open␣file␣to␣write␣trails.\n"
    );
break;
}
else
{
if((write_success_fp = fopen("C:/Users/
    Jaime/Documents/ESCOM_SEMESTRE_4/2
    CM5_TEORIA_COMPUTACIONAL/UNIT_2/CHESS/
    Successful_Trails.txt", "w")) == NULL)
{
printf("Can't␣open␣file␣to␣write␣successful
    ␣trails.\n");
break;
}
printf("\n|+|+|The␣string␣is:␣%s\n", string
    );
fprintf(write_trails_fp, "\n|+|+|The␣string
    ␣is:␣%s\n", string);
fprintf(write_success_fp, "\n|+|+|The␣
    string␣is:␣%s\n", string);
printf("|+|+|The␣last␣level␣is:␣%d\n",
    levels);
fprintf(write_trails_fp, "|+|+|The␣last␣
    level␣is:␣%d\n", levels);
fprintf(write_success_fp, "|+|+|The␣last␣
    level␣is:␣%d\n", levels);
printf("|+|+|The␣number␣of␣nines␣is:␣%d\n",
     nines);
fprintf(write_trails_fp, "|+|+|The␣number␣
    of␣nines␣is:␣%d\n", nines);
fprintf(write_success_fp, "|+|+|The␣number␣
    of␣nines␣is:␣%d\n", nines);
```

```
printf("|+|+|The number of nodes is: %d\n",
     nodes);
fprintf(write_trails_fp, "|+|+|The number
    of nodes is: %d\n", nodes);
fprintf(write_success_fp, "|+|+|The number
    of nodes is: %d\n", nodes);
if((position = SearchNode(&tree, '9',
    levels)) != NULL) //Search if any node
    with '9' exists ergo the string
    succeded
{
printf("|+|+|The string succeeded.\n\n");
fprintf(write_trails_fp, "|+|+|The string
    succeeded.\n\n");
fprintf(write_success_fp, "|+|+|The string
    succeeded.\n\n");
}
else
{
printf("|+|+|The string did not succeed.\n\
    n");
fprintf(write_trails_fp, "|+|+|The string
    did not succeed.\n\n");
fprintf(write_success_fp, "|+|+|The string
    did not succeed.\n\n");
}
for(position = Root(&tree); position !=
    NULL && nodes > 0; position = Root(&
    tree), nodes--) //Looks for all nodes
    in the specified level
{
if((position = SearchNode(&tree, 'A',
    levels)) != NULL) //The code 'A' means
    searching for All nodes in the
    specified level, in this case in the
    last level
{
if(position->e.c == '9' && successfuls > 0)
     //If the node contains a '9' it means
    this trail is succesful so we print it
    to the corresponding file
{
printf("Found node %c in level %d.\n",
    position->e.c, position->e.level);
fprintf(write_success_fp, "Found node %c in
    level %d.\n", position->e.c, position
    ->e.level);
position->e.level = -1;
for(i = 0; position != NULL; position =
    Father(&tree, position))
{
trace[i++] = position->e.c;
}
printf("The trail is: ");
fprintf(write_success_fp, "The trail is: ")
    ;
for(j = 0; j < i; j++)
```

```
                        {
                        printf("%c--", trace[j]);
                        fprintf(write_success_fp, "%c--", trace[j])
                            ;
                        }
                        printf("\n\n");
                        fprintf(write_success_fp, "\n\n");
                        successfuls--; //The condition specifies
                            that we only search for 3 successful
                            trails
                        }
                        else //Here the program prints all trails
                            of the nodes in the last level
                        {
                        printf("Found node %c in level %d.\n",
                            position->e.c, position->e.level);
                        fprintf(write_trails_fp, "Found node %c in
                            level %d.\n", position->e.c, position->
                            e.level);
                        position->e.level = -1;
                        for(i = 0; position != NULL; position =
                            Father(&tree, position))
                        {
                        trace[i++] = position->e.c;
                        }
                        printf("The trail is: ");
                        fprintf(write_trails_fp, "The trail is: ");
                        for(j = 0; j < i; j++)
                        {
                        printf("%c--", trace[j]);
                        fprintf(write_trails_fp, "%c--", trace[j]);
                        }
                        printf("\n\n");
                        fprintf(write_trails_fp, "\n\n");
                        }
                        }
                        }
                        }
                        fclose(write_trails_fp);
                        fclose(write_success_fp);
                }
        }
        return 0;
}
```

3.3.1. Código del TADArbol.h

```
#include <stdbool.h>

typedef struct
{
        char c;
        bool visited;
        int level;
        //Lo que gustes
}element;
```

```
typedef struct node
{
        element e;
        struct node *left;
        struct node *right;
        struct node *middle_right;
        struct node *middle_left;
}node;

typedef node *Tree;

typedef node *Position;

void Initialize(Tree *t); //*
void Destroy(Tree *t); //*
Position Root(Tree *t); //*
Position Father(Tree *t, Position p); //*
Position RightSon(Position p); //*
Position LeftSon(Position p);//*
Position Search(Tree *t, element e);//*
bool Empty(Tree *t);//*
bool NullNode(Tree *t, Position p);//
element ReadNode(Position p);
void NewRightSon(Tree *t, Position p, element e); //*
void NewLeftSon(Tree *t, Position p, element e); //*
void DeleteRightSon(Position p); //*
void DeleteLeftSon(Position p); //*
void DeleteNode(Tree *t, Position p); //*
void ReplaceNode(Position p, element e);//*

//RECENTLY ADDED
Position NextSon(Position father, Position position, bool *
    son_visited);
void NewMiddleRightSon(Tree *t, Position p, element e);
void NewMiddleLeftSon(Tree *t, Position p, element e);
Position SearchNonVisited(Tree *t, int levels);
Position SearchInLevel(Tree *t, int levels);
Position SearchNode(Tree *t, char ch, int level);
bool IsSon(Position father, Position position);
void PreOrder(Tree *t, Position position);
Position MiddleRightSon(Position position);
Position MiddleLeftSon(Position position);
```

3.3.2. Código del TADArbol.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "TADArbol.h"

void Initialize(Tree *t)
{
        *t = NULL;

        return;
```

```
}

void Destroy(Tree *t)
{
        if((*t)->right != NULL)
                Destroy(&((*t)->right));
        if((*t)->left != NULL)
                Destroy(&((*t)->left));

        free(*t);

        return;
}

void NewRightSon(Tree *t, Position p, element e)
{
        if(Empty(t))
        {
                *t = malloc(sizeof(node));
                (*t)->e = e;
                (*t)->right = NULL;
                (*t)->left = NULL;
                (*t)->middle_right = NULL;
                (*t)->middle_left = NULL;
        }
        else
        {
                p->right = malloc(sizeof(node));
                p->right->e = e;
                p->right->right = NULL;
                p->right->left = NULL;
                p->right->middle_right = NULL;
                p->right->middle_left = NULL;
        }

        return;
}

void NewLeftSon(Tree *t, Position p, element e)
{
        if(Empty(t))
        {
                *t = malloc(sizeof(node));
                (*t)->e = e;
                (*t)->right = NULL;
                (*t)->left = NULL;
                (*t)->middle_right = NULL;
                (*t)->middle_left = NULL;
        }
        else
        {
                p->left = malloc(sizeof (node));
                p->left->e = e;
                p->left->right = NULL;
                p->left->left = NULL;
                p->left->middle_right = NULL;
                p->left->middle_left = NULL;
```

```
        }

        return;
}

void DeleteRightSon(Position p)
{
        Destroy(&(p->right));
        p->right = NULL;
        return;
}

void DeleteLeftSon(Position p)
{
        Destroy(&(p->left));
        p->left = NULL;
        return;
}

void DeleteNode(Tree *t, Position p)
{
        Position father = Father(t, p);

        if (father->right == p)
                father->right = NULL;
        else
                if (father->left == p)
                        father->left = NULL;

        Destroy(&p);

        return;
}

void ReplaceNode(Position p, element e)
{
    p->e = e;

        return;
}

Position Root(Tree *t)
{
        return *t;
}

Position Father(Tree *t, Position p)
{
        Position father = NULL;

        if((*t)->right == p || (*t)->middle_right == p || (*t)->
            middle_left == p || (*t)->left == p)
                return *t;

        if((*t)->right != NULL)
                father = Father(&((*t)->right), p);
```

```
        if((*t)->middle_right != NULL && father == NULL)
                father = Father(&((*t)->middle_right), p);

        if((*t)->middle_left != NULL && father == NULL)
                father = Father(&((*t)->middle_left), p);

        if((*t)->left != NULL && father == NULL)
                father = Father(&((*t)->left), p);

        return father;
}

Position RightSon(Position p)
{
        return p->right;
}

Position LeftSon(Position p)
{
        return p->left;
}

Position Search(Tree *t, element e)
{
        Position p = NULL;

        if (memcmp(&((*t)->e), &e, sizeof(element)) == 0)
                return *t;
        if((*t)->right != NULL)
                p = Search(&((*t)->right), e);
        if((*t)->left != NULL && p == NULL)
                p = Search(&((*t)->left), e);
        if((*t)->middle_left != NULL && p == NULL)
                p = Search(&((*t)->middle_left), e);
        if((*t)->middle_right != NULL && p == NULL)
                p = Search(&((*t)->middle_right), e);

        return p;
}


bool Empty(Tree *t)
{
        if(*t != NULL)
                return false;
        else
                return true;
}

bool NullNode(Tree *t, Position position)
{
        bool b = true;

        if (*t == position)
                return false;
        if((*t)->right != NULL)
                b = NullNode(&((*t)->right), position);
```

```
        if((*t)->middle_right != NULL && b == true)
                b = NullNode(&((*t)->middle_right), position);
        if((*t)->middle_left != NULL)
                b = NullNode(&((*t)->middle_left), position);
        if((*t)->left != NULL && b == true)
                b = NullNode(&((*t)->left), position);

        return b;
}

element ReadNode(Position p)
{
  return p->e;
}

//|+|+|+|+|+|+|+|+|+|RECENTLY CREATED
Position NextSon(Position father, Position position, bool *
    son_visited)
{
        if(father->middle_right != position && son_visited[0] !=
            true)
        {
                son_visited[0] = true;
                return father->middle_right;
        }
        else if(father->middle_left != position && son_visited[1]
            != true)
        {
                son_visited[1] = true;
                return father->middle_left;
        }
        else if(father->left != position && son_visited[2] != true)
        {
                son_visited[2] = true;
                return father->left;
        }

        return NULL;
}

Position SearchNonVisited(Tree *t, int levels)
{
        Position position = NULL;

        if ((*t)->e.visited == false && (*t)->e.level == levels)
                return *t;
        if((*t)->right != NULL)
                position = SearchNonVisited(&((*t)->right), levels)
                    ;
        if((*t)->middle_right != NULL && position == NULL)
                position = SearchNonVisited(&((*t)->middle_right),
                    levels);
        if((*t)->middle_left != NULL && position == NULL)
                position = SearchNonVisited(&((*t)->middle_left),
                    levels);
        if((*t)->left != NULL && position == NULL)
                position = SearchNonVisited(&((*t)->left), levels);
```

```
        return position;
}

Position SearchInLevel(Tree *t, int levels)
{
        Position position = NULL;

        if ((*t)->e.level == levels)
                return *t;

        if((*t)->right != NULL)
                position = SearchInLevel(&((*t)->right), levels);

        if((*t)->middle_right != NULL && position == NULL)
                position = SearchInLevel(&((*t)->middle_right),
                    levels);

        if((*t)->middle_left != NULL && position == NULL)
                position = SearchInLevel(&((*t)->middle_left),
                    levels);

        if((*t)->left != NULL && position == NULL)
                position = SearchInLevel(&((*t)->left), levels);

        return position;
}

void NewMiddleRightSon(Tree *t, Position p, element e)
{
        if(Empty(t))
        {
                *t = malloc(sizeof(node));
                (*t)->e = e;
                (*t)->right = NULL;
                (*t)->left = NULL;
                (*t)->middle_right = NULL;
                (*t)->middle_left = NULL;
        }
        else
        {
                p->middle_right = malloc(sizeof(node));
                p->middle_right->e = e;
                p->middle_right->right = NULL;
                p->middle_right->left = NULL;
                p->middle_right->middle_right = NULL;
                p->middle_right->middle_left = NULL;
        }

        return;
}

void NewMiddleLeftSon(Tree *t, Position p, element e)
{
        if(Empty(t))
        {
                *t = malloc(sizeof(node));
```

```
                (*t)->e = e;
                (*t)->right = NULL;
                (*t)->left = NULL;
                (*t)->middle_right = NULL;
                (*t)->middle_left = NULL;
        }
        else
        {
                p->middle_left = malloc(sizeof (node));
                p->middle_left->e = e;
                p->middle_left->right = NULL;
                p->middle_left->left = NULL;
                p->middle_left->middle_right = NULL;
                p->middle_left->middle_left = NULL;
        }

        return;
}

bool IsSon(Position father, Position position)
{
        if(father->right == position || father->middle_right ==
            position || father->middle_left == position || father->
            left == position)
                return true;

        return false;
}

Position MiddleRightSon(Position position)
{
        return position->middle_right;
}

Position MiddleLeftSon(Position position)
{
        return position->middle_left;
}

void PreOrder(Tree *t, Position position)
{
        element e;
        if(!NullNode(t, position))
        {
                e = ReadNode(position);
                printf(" %c", e.c);
                PreOrder(t, LeftSon(position));
                PreOrder(t, MiddleLeftSon(position));
                PreOrder(t, MiddleRightSon(position));
                PreOrder(t, RightSon(position));
        }
        return;
}

Position SearchNode(Tree *t, char ch, int levels)
{
        Position p = NULL;
```

```
        if(ch == 'A')
        {
                if((*t)->e.level == levels)
                        return *t;
        }
        else if((*t)->e.level == levels && (*t)->e.c == ch)
        {
                return *t;
        }

        if((*t)->right != NULL)
                p = SearchNode(&((*t)->right), ch, levels);

        if((*t)->middle_right != NULL && p == NULL)
                p = SearchNode(&((*t)->middle_right), ch, levels);

        if((*t)->middle_left != NULL && p == NULL)
                p = SearchNode(&((*t)->middle_left), ch, levels);

        if((*t)->left != NULL && p == NULL)
                p = SearchNode(&((*t)->left), ch, levels);

        return p;
}
```

4. Práctica 3: Expresión regular $(0+10)^*(\epsilon+1)$.

4.1. Descripción

Este programa genera 5 cadenas aleatorias dadas por la expresión regular $(0+10)^*(\epsilon+1)$, el programa muestra en consola cual fue la longitud random de cada cadena con un máximo de 1000 símbolos además de mandar a un archivo las cadenas.

El programa también cuenta con una opción para mostrar el archivo en el cual se encuentran las cadenas.

4.2. Ejecución

Al iniciar el programa se nos presenta el menú con el siguiente aspecto:

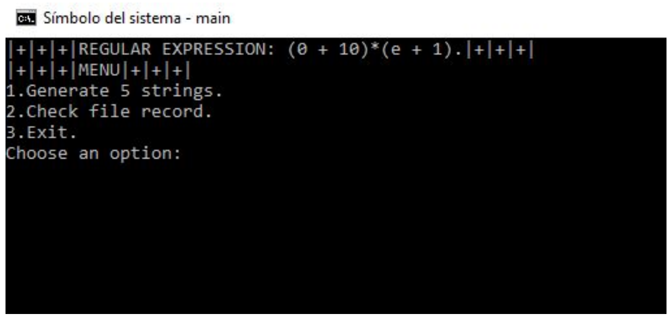


Figura 14: Menú Principal

Si elegimos la opción 1, el programa desplegará las 5 cadenas en consola junto con sus longitudes, además de mostrar las cadenas en un archivo:

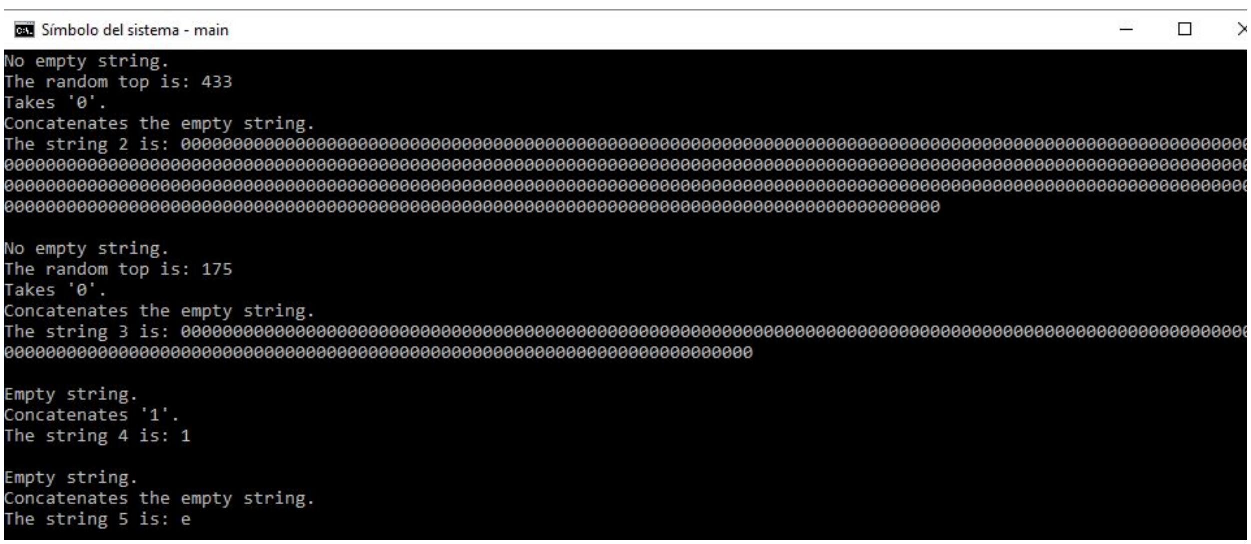


Figura 15: Opción 1

Si elegimos la opción 2, el programa nos mostrará el arcihvo con las 5 cadenas:

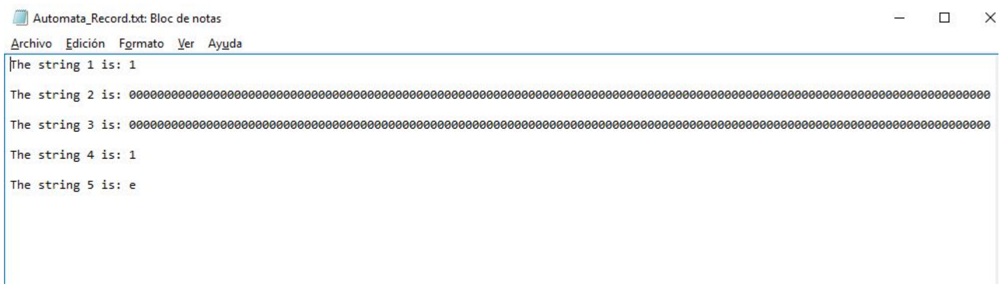


Figura 16: Opción 2

4.3. Código

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

#define RANDOM_TOP 1000

int main(void)
{
        int option = 0, k = 0, i = 0, j = 0;
        char *string;
        FILE *write_fp;

        srand((unsigned) time(NULL));
        for(;;)
        {
                printf("|+|+|+|REGULAR EXPRESSION: (0 + 10)*(e + 1)
                    .|+|+|+|\n");
                printf("|+|+|+|MENU|+|+|+|\n");
                printf("1. Generate 5 strings.\n");
                printf("2. Check file record.\n");
                printf("3. Exit.\n");
                printf("Choose an option: ");
                scanf(" %d", &option);

                switch(option)
                {
                        case 1:
                        system("cls");
                        if((write_fp = fopen("C:/Users/Jaime/
                            Documents/ESCOM_SEMESTRE_4/2
                            CM5_TEORIA_COMPUTACIONAL/UNIT_2/RE_1/
                            Automata_Record.txt", "w")) == NULL)
                        {
                        printf("Can't open Automata_record.txt\n");
                        break;
                        }
                        for(j = 0; j < 5; j++)
                        {
                        //Computes in case of epsilon/empty string
                        if((rand() % 2) == 1) //If 1 we don't add
                            any symbol to the string, that means
                            the empty string
                        {
                        printf("Empty string.\n");
                        //Computes (e + 1)
                        if((rand() % 2) == 1) //If 1 we concatenate
                             the string with nothing, that means we
                             concatenate with epsilon
                        {
                        printf("Concatenates the empty string.\n");
                        string = malloc(2); //C guarantees that a
                            char only occupies one byte so we can
                            write 1 instead of sizeof(char)
                        string[0] = 'e';
```

```
string[1] = '\0';
printf("The string %d is: %s\n\n", j + 1,
    string);
fprintf(write_fp, "The string %d is: %s\n\n
    ", j + 1, string);
}
else     //We concatenate with 1
{
printf("Concatenates '1'.\n");
string = malloc(2);
string[0] = '1';
string[1] = '\0';
printf("The string %d is: %s\n\n", j + 1,
    string);
fprintf(write_fp, "The string %d is: %s\n\n
    ", j + 1, string);
}
}
//Computes (0 + 10)* when there is no
    epsilon/empty string
else
{
printf("No empty string.\n");
k = rand() %RANDOM_TOP + 1;
printf("The random top is: %d\n", k);
if((rand() % 2) == 1) //If 1 generates
    symbol '0' k-times
{
printf("Takes '0'.\n");
string = malloc(k + 2); //Adds 2 because of
     the next symbol to concatenate and
    plus the '\0' character at the end
for(i = 0; i < k; i++)
        string[i] = '0';
}
else
{
printf("Takes \"10\".\n");
string = malloc((2 * k) + 2); //Alocates
    memory for the two simbols "10" plus
    the next symbol in concatenation plus
    the '\0' character
for(i = 0; i < 2 * k; i++)
{
        string[i++] = '1';
        string[i] = '0';
}
}
//Computes (e + 1)
if((rand() % 2) == 1) //Concatenates
    epsilon, that means nothing
{
printf("Concatenates the empty string.\n");
string[i] = '\0';
printf("The string %d is: %s\n\n", j + 1,
    string);
```

```
                        fprintf(write_fp, "The string %d is: %s\n\n
                            ", j + 1, string);
                        }
                        else //Concatenates '1'
                        {
                        printf("Concatenates '1'.\n");
                        string[i++] = '1';
                        string[i] = '\0';
                        printf("The string %d is: %s\n\n", j + 1,
                            string);
                        fprintf(write_fp, "The string %d is: %s\n\n
                            ", j + 1, string);
                        }
                        }
                        }
                        fclose(write_fp);
                        break;
                        case 2:
                        system("C:/Users/Jaime/Documents/
                            ESCOM_SEMESTRE_4/2
                            CM5_TEORIA_COMPUTACIONAL/UNIT_2/RE_1/
                            Automata_Record.txt");
                        break;
                        case 3:
                        system("cls");
                        exit(EXIT_SUCCESS);
                        break;
                        default:
                        system("cls");
                        printf("Choose a correct option.\n");
                        Sleep(2000);
                        break;
                }
        }
        return 0;
}
```

5. Práctica 4: Expresión regular $[(10)^*0+1(01)^*1][(0(01)^*(1+00)+1(10)^*(0+11))]^*$.

5.1. Descripción

Este programa genera 5 cadenas aleatorias dadas por la expresión regular $[(10)^*0+1(01)^*1][(0(01)^*(1+00)+1(10)^*(0+11))]^*$, el programa muestra en consola cual fue la longitud random de cada cadena con un máximo de 1000 símbolos además de mandar a un archivo las cadenas.

El programa también cuenta con una opción para mostrar el archivo en el cual se encuentran las cadenas.

5.2. Ejecución

Al iniciar el programa se nos presenta el menú con el siguiente aspecto:

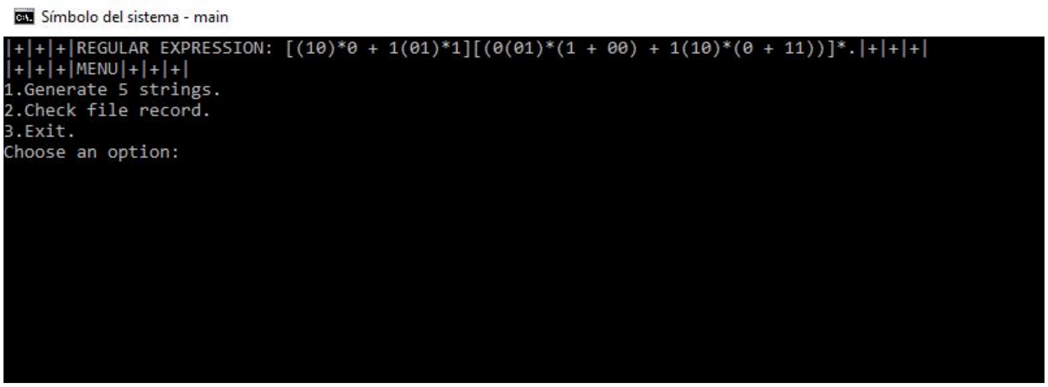


Figura 17: Menú Principal

Si elegimos la opción 1, el programa desplegará las 5 cadenas en consola junto con sus longitudes, además de mostrar las cadenas en un archivo:

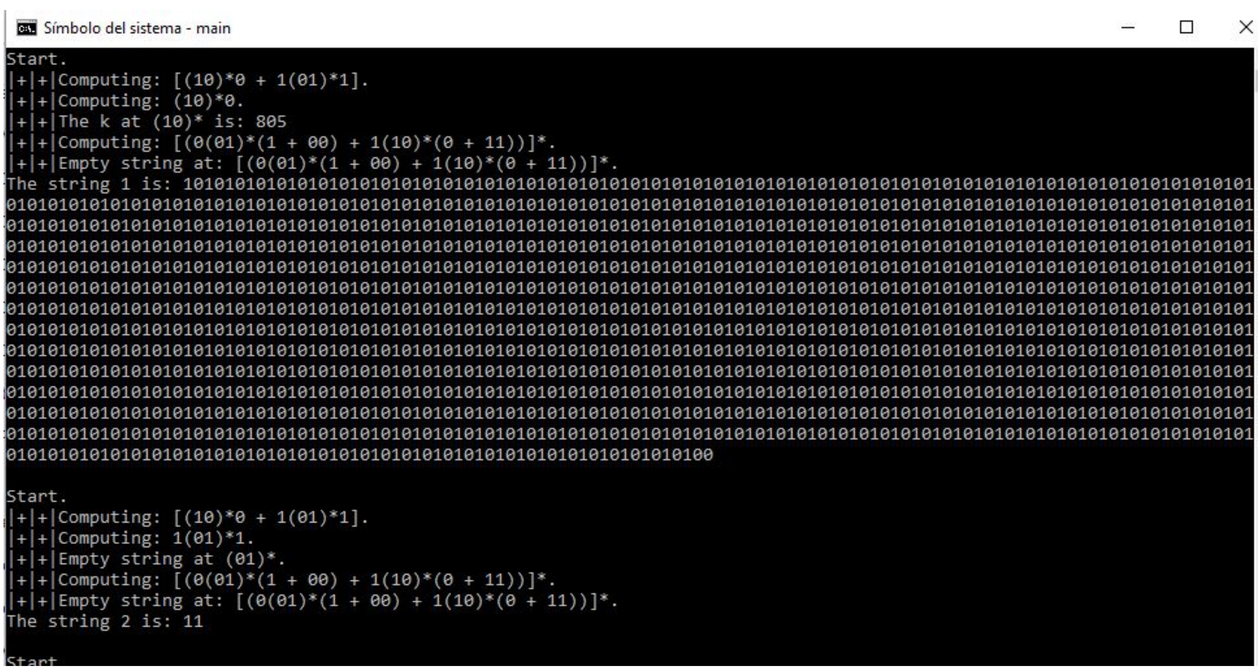


Figura 18: Opción 1

Si elegimos la opción 2, el programa nos mostrará el arcihvo con las 5 cadenas:

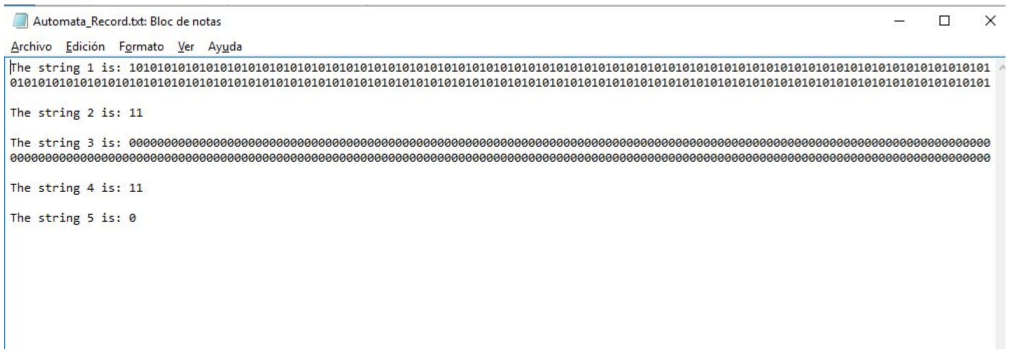


Figura 19: Opción 2

5.3. Código

```
#include <stdio.h>
#include <windows.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>

#define RANDOM_TOP 1000

int main(void)
{
        int option = 0, k = 0, i = 0, j = 0;
        long int length = 0;
        int index;
        char *string1, *alpha_string;
        FILE *write_fp;

        srand((unsigned) time(NULL));
        for(;;)
        {
                printf("|+|+|+|REGULAR EXPRESSION: [(10)*0 + 1(01)
                    *1][(0(01)*(1 + 00) + 1(10)*(0 + 11))
                    ]*.|+|+|+|\n");
                printf("|+|+|+|MENU|+|+|+|\n");
                printf("1. Generate 5 strings.\n");
                printf("2. Check file record.\n");
                printf("3. Exit.\n");
                printf("Choose an option: ");
                scanf(" %d", &option);

                switch(option)
                {
                        case 1:
                        system("cls");
                        if((write_fp = fopen("C:/Users/Jaime/
                            Documents/ESCOM_SEMESTRE_4/2
                            CM5_TEORIA_COMPUTACIONAL/UNIT_2/RE_2/
                            Automata_Record.txt", "w")) == NULL)
                        {
                        printf("Can't open Automatomata_record.txt\
                            n");
                        break;
                        }
                        for(j = 0; j < 5; j++)
                        {
                        printf("Start.\n");
                        index = 0;
                        length = 0;
                        k = 0;
                        i = 0;
                        printf("|+|+|Computing: [(10)*0 + 1(01)
                            *1].\n");
                        if((rand() % 2) == 1)
                        {
                        printf("|+|+|Computing: (10)*0.\n");
                        if((rand() % 2) == 1)
```

```
{
k = rand() %RANDOM_TOP + 1;
printf("|+|+|The k at (10)* is: %d\n", k);
length = k * 2;
string1 = calloc(length, sizeof(char));
for(i = index; i < length; i++)
{
        string1[i++] = '1';
        string1[i] = '0';
}
index = i;
length++;
string1 = realloc(string1, length);
string1[index++] = '0';
}
else
{
printf("|+|+|Empty string at (10)*.\n");
length++;
string1 = calloc(length, sizeof(char));
string1[index++] = '0';
length = index;
}
}
else
{
printf("|+|+|Computing: 1(01)*1.\n");
length++;
string1 = calloc(length, sizeof(char));
string1[index++] = '1';
if((rand() % 2) == 1)
{
k = rand() %RANDOM_TOP + 1;
printf("|+|+|The k at (01)* is: %d\n", k);
length = length + (k * 2);
string1 = realloc(string1, length);
for(i = index; i < length; i++)
{
        string1[i++] = '0';
        string1[i] = '1';
}
index = i;
length++;
string1 = realloc(string1, length);
string1[index++] = '1';
}
else
{
printf("|+|+|Empty string at (01)*.\n");
length++;
string1 = realloc(string1, length);
string1[index++] = '1';
}
}
printf("|+|+|Computing: [(0(01)*(1 + 00) +
    1(10)*(0 + 11))]*.\n");
if((rand() % 2) == 1)
```

```
{
printf("|+|+|Empty string at: [(0(01)*(1 + 00) + 1(10)*(0 + 11))]*.\n");
length++;
string1 = realloc(string1, length);
string1[index] = '\0';
printf("The string %d is: %s\n\n", j + 1, string1);
fprintf(write_fp, "The string %d is: %s\n\n", j + 1, string1);
}
else
{
if((rand() % 2) == 1)
{
printf("|+|+|Computing: 0(01)*(1 + 00).\n");
length++;
string1 = realloc(string1, length);
string1[index++] = '0';
if((rand() % 2) == 1)
{
k = rand() %RANDOM_TOP + 1;
printf("|+|+|The k at (01)* is: %d\n", k);
length = length + (k * 2);
string1 = realloc(string1, length);
for(i = index; i < length; i++)
{
        string1[i++] = '0';
        string1[i] = '1';
}
index = i;
}
else
printf("|+|+|Empty string at (01)*.\n");
if((rand() % 2) == 1)
{
length += 2;
string1 = realloc(string1, length);
string1[index++] = '1';
string1[index] = '\0';
}
else
{
length += 3;
string1 = realloc(string1, length);
string1[index++] = '0';
string1[index++] = '0';
string1[index] = '\0';
}
}
else
{
printf("|+|+|Computing: 1(10)*(0 + 11)\n");
length++;
string1 = realloc(string1, length);
string1[index++] = '1';
```

```
if((rand() % 2) == 1)
{
k = rand() %RANDOM_TOP + 1;
printf("|+|+|The k at (10)* is: %d\n", k);
length = length + (k * 2);
string1 = realloc(string1, length);
for(i = index; i < length; i++)
{
        string1[i++] = '1';
        string1[i] = '0';
}
index = i;
}
else
printf("|+|+|Empty string at (10)*.\n");
if((rand() % 2) == 1)
{
length += 2;
string1 = realloc(string1, length);
string1[index++] = '0';
string1[index] = '\0';
}
else
{
length += 3;
string1 = realloc(string1, length);
string1[index++] = '1';
string1[index++] = '1';
string1[index] = '\0';
}
}
k = rand() %RANDOM_TOP + 1;
printf("The k at [(0(01)*(1 + 00) + 1(10)
    *(0 + 11))]* is: %d\n", k);
length = strlen(string1);
printf("The length of the string is: %d\n"
    , length);
alpha_string = malloc((length * k) + 1);
strcpy(alpha_string, string1);
for(i = 0; i < (k - 1); i++)
        strcat(alpha_string, string1);
printf("The string %d is: %s\n\n", j + 1,
    alpha_string);
fprintf(write_fp, "The string %d is: %s\n\n
    ", j + 1, alpha_string);
}
free(string1);
free(alpha_string);
}
fclose(write_fp);
break;
case 2:
system("C:/Users/Jaime/Documents/
    ESCOM_SEMESTRE_4/2
    CM5_TEORIA_COMPUTACIONAL/UNIT_2/RE_2/
    Automata_Record.txt");
break;
```

```
                        case 3:
                        system("cls");
                        exit(EXIT_SUCCESS);
                        break;
                        default:
                        system("cls");
                        printf("Choose a correct option.\n");
                        Sleep(2000);
                        break;
                }
        }
        return 0;
}
```

6. Práctica 5: Gramática Independiente del Contexto (Palíndromo).

6.1. Descripción

Las reglas que denen los palíndromos, expresadas empleando la notación de la gramática independiente del contexto, se muestran a continuación:

1. $P \rightarrow \epsilon$
2. $P \rightarrow 0$
3. $P \rightarrow 1$
4. $P \rightarrow 0P0$
5. $P \rightarrow 1P1$

Las tres primeras reglas denen el caso básico. Establecen que la clase de palíndromos incluye las cadenas $, 0y1$. Ninguno de los lados de la derecha de estas reglas(la parte que sigue a las echas)contiene una variable, razón por la que constituyen el caso básico de la denición. Las dos últimas reglas forman la parte inductiva de la denición. Por ejemplo, la regla 4 establece que si tomamos cualquier cadena w de la clase P, entonces 0w0 también pertenece a la clase P. Del mismo modo, la regla 5 nos dice que 1w1 también pertenece a P.

Este programa genera una cadena de 0's y 1's que tiene la propiedad de ser un palíndromo, con la longitud específicada por el usuario, el programa imprime en consola y en archivo, las reglas aleatorias que seleccionó el programa para generar la cadena así como la cadena. Cuenta con la opción de mostrar el archivo en el cual se generó el historial de reglas y la cadena, y otra opción para que solo muestre la cadena para poder apreciar la forma que toma la cadena conforme va creciendo.

6.2. Ejecución

Al iniciar el programa se nos presenta el menú con el siguiente aspecto:

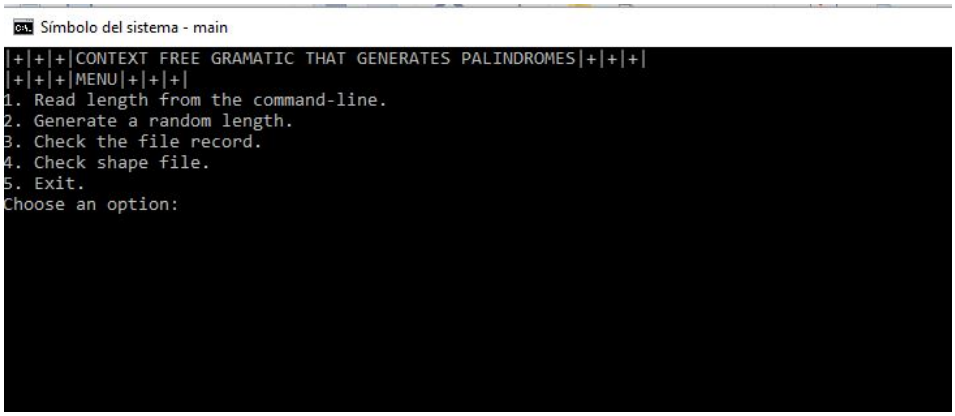


Figura 20: Menú Principal

Si elegimos la opción 1, el programa nos pedirá ingresar la longitud deseada del palíndromo y desplegará las reglas que se fueron tomando junto con su cadena respectiva:

Figura 21: Opción 1

Si elegimos la opción 2, el programa generará una longitud aleatoria (menor o igual a 100,000)y desplegará las reglas que se fueron tomando junto con su cadena respectiva y la longitud aleatoria:

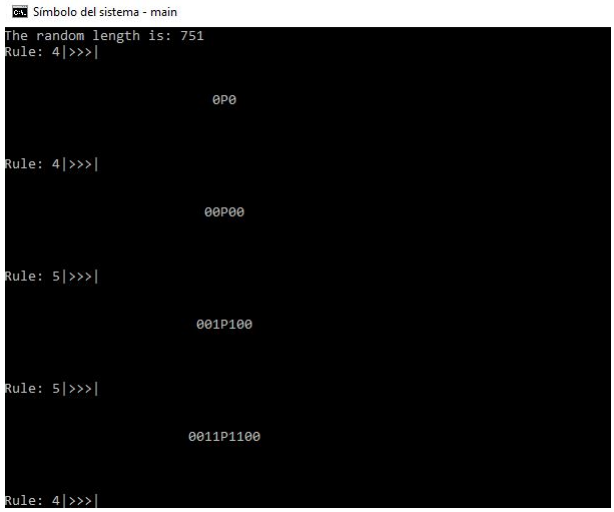


Figura 22: Opción 2

Si elegimos la opción 3, el programa mostrará el archivo en cual se guardó el historial de reglas junto con las cadenas respectivas:

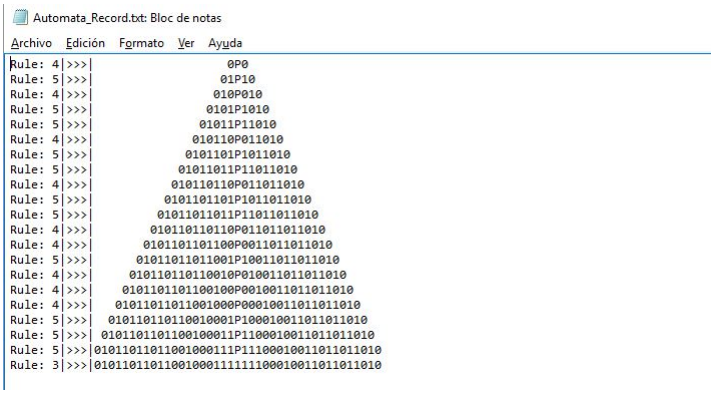


Figura 23: Opción 3

Si elegimos la opción 4, el programa abrirá el archivo en el cual se encuentran solo las cadenas:

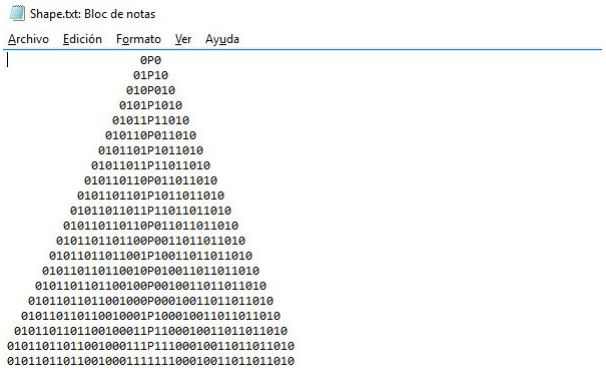


Figura 24: Opción 4

6.3. Código

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>
#include <string.h>

#define RULE4 4
#define RULE5 5
#define RANDOM_TOP 1000

bool first_time[3] = {true};

char *upToP(char *string, int length, int rule);
char *fromP(char *string, int length, int rule);

int main(void)
{
        int option = 0, rule;
        long int length = 0, index = 0, choices = 0, i = 0, j = 0,
            k = 0, random = 0;
        char *string, *aux_string, *prev_string, *auxe_string;
        FILE *write_fp, *write_shape;

        srand((unsigned) time(NULL));
        for(;;)
        {
                length = 0;
                index = 0;
                choices = 0;
                printf("|+|+|+|CONTEXT_FREE_GRAMATIC_THAT_GENERATES
                    _PALINDROMES|+|+|+|\n");
                printf("|+|+|+|MENU|+|+|+|\n");
                printf("1. Read length from the command-line.\n");
                printf("2. Generate a random length.\n");
                printf("3. Check the file record.\n");
                printf("4. Check shape file.\n");
                printf("5. Exit.\n");
                printf("Choose an option: ");
                scanf(" %d", &option);
                switch(option)
                {
                        case 1: case 2:
                        system("cls");
                        write_fp = fopen("C:/Users/Jaime/Documents/
                            ESCOM_SEMESTRE_4/2
                            CM5_TEORIA_COMPUTACIONAL/UNIT_2/CFG_1/
                            Automata_Record.txt", "w");
                        write_shape = fopen("C:/Users/Jaime/
                            Documents/ESCOM_SEMESTRE_4/2
                            CM5_TEORIA_COMPUTACIONAL/UNIT_2/CFG_1/
                            Shape.txt", "w");
                        if(option == 1)
                        {
                        printf("Enter the length: ");
```

```
scanf("%d", &length);
}
else
{
length = rand() %RANDOM_TOP;
printf("The random length is: %d\n", length
    );
fprintf(write_fp, "The random length is: %d
    \n", length);
fprintf(write_shape, "The random length is:
    %d\n", length);
}
if(length == 0)
{
printf("Rule: 1|>>>|P --> e.\n");
fprintf(write_fp, "Rule: 1|>>>|P --> e.\n")
    ;
fprintf(write_shape, "Rule: 1|>>>|P --> e.\
    n");
}
else if(length == 1)
{
if((rand() % 2) == 0)
{
printf("Rule: 2|>>>|P --> 0.\n");
fprintf(write_fp, "Rule: 2|>>>|P --> 0.\n")
    ;
fprintf(write_shape, "Rule: 2|>>>|P --> 0.\
    n");
}
else
{
printf("Rule: 2|>>>|P --> 1.\n");
fprintf(write_fp, "Rule: 2|>>>|P --> 1.\n")
    ;
fprintf(write_shape, "Rule: 2|>>>|P --> 1.\
    n");
}
}
else
{
if(length % 2 == 0)
        length++;
string = malloc(length + 1);
for(i = 0; i < length; i++)
        string[i] = ' ';
string[i] = '\0';
string[length / 2] = 'P';
choices = length / 2;
for(i = 0; i < choices; i++)
{
if((rand() % 2) == 1)
        rule = RULE4;
else
        rule = RULE5;

aux_string = upToP(string, length, rule);
```

```
for(j = 0; string[j] != 'P'; j++)
        string[j] = aux_string[j];
aux_string = fromP(string, length, rule);
for(j = (length / 2) + 1, k = 0; string[j]
    != '\0'; j++, k++)
        string[j] = aux_string[k];

printf("Rule: %d|>>>|%s\n", rule, string);
fprintf(write_fp, "Rule: %d|>>>|%s\n", rule
    , string);
fprintf(write_shape, "%s\n", string);
}
random = rand() % 3;
if(random == 0)
{
aux_string = malloc(length + 1);
for(i = 0; string[i] != 'P'; i++)
        aux_string[i] = string[i];
for(; string[i] != '\0'; i++)
        aux_string[i] = string[i + 1];
aux_string[i] = '\0';
printf("Rule: 1|>>>| %s\n", aux_string);
fprintf(write_fp, "Rule: 1|>>>| %s\n",
    aux_string);
fprintf(write_shape, " %s\n", aux_string);
}
else if(random == 1)
{
string[length / 2] = '0';
printf("Rule: 2|>>>|%s\n", string);
fprintf(write_fp, "Rule: 2|>>>|%s\n",
    string);
fprintf(write_shape, "%s\n", string);
}
else if(random == 2)
{
string[length / 2] = '1';
printf("Rule: 3|>>>|%s\n", string);
fprintf(write_fp, "Rule: 3|>>>|%s\n",
    string);
fprintf(write_shape, "%s\n", string);
}
fclose(write_fp);
fclose(write_shape);
}
break;
case 3:
system("C:/Users/Jaime/Documents/
    ESCOM_SEMESTRE_4/2
    CM5_TEORIA_COMPUTACIONAL/UNIT_2/CFG_1/
    Automata_Record.txt");
system("cls");
break;
case 4:
system("C:/Users/Jaime/Documents/
    ESCOM_SEMESTRE_4/2
    CM5_TEORIA_COMPUTACIONAL/UNIT_2/CFG_1/
```

```
                                Shape.txt");
                        system("cls");
                        break;
                        case 5:
                        system("cls");
                        exit(EXIT_SUCCESS);
                        break;
                        default:
                        system("cls");
                        printf("Choose a correct option.\n");
                        Sleep(2000);
                        break;
                }
        }
        return 0;
}

char *upToP(char *string, int length, int rule)
{
        long int i, aux_length;
        char *aux_string;

        aux_string = malloc((length / 2) + 1);
        for(i = 0; string[i] != 'P'; i++)
                aux_string[i] = string[i];
        aux_string[i] = '\0';

        if(first_time[0] == true && rule == RULE4)
        {
                aux_string[(length / 2) - 1] = '0';
                first_time[0] = false;
                return aux_string;
        }
        else if(first_time[0] == true && rule == RULE5)
        {
                aux_string[(length / 2) - 1] = '1';
                first_time[0] = false;
                return aux_string;
        }

        if(first_time[0] == false && rule == RULE4)
        {
                aux_length = strlen(aux_string);
                for(i = 0; i < (aux_length - 1); i++)
                        aux_string[i] = aux_string[i + 1];
                aux_string[aux_length - 1] = '0';
        }
        else if(first_time[0] == false && rule == RULE5)
        {
                aux_length = strlen(aux_string);
                for(i = 0; i < (aux_length - 1); i++)
                        aux_string[i] = aux_string[i + 1];
                aux_string[aux_length - 1] = '1';
        }

        return aux_string;
}
```

```
char *fromP(char *string, int length, int rule)
{
        long int i, j, aux_length;
        char *aux_string;

        aux_string = malloc((length / 2) + 1);
        for(i = (length / 2) + 1, j = 0; string[i] != '\0'; i++, j
            ++)
                aux_string[j] = string[i];
        aux_string[j] = '\0';

        if(first_time[1] == true && rule == RULE4)
        {
                aux_string[0] = '0';
                first_time[1] = false;
                return aux_string;
        }
        else if(first_time[1] == true && rule == RULE5)
        {
                aux_string[0] = '1';
                first_time[1] = false;
                return aux_string;
        }

        if(first_time[1] == false && rule == RULE4)
        {
                aux_length = strlen(aux_string);
                for(i = aux_length; i > 0; i--)
                        aux_string[i] = aux_string[i - 1];
                aux_string[i] = '0';
                aux_string = realloc(aux_string, aux_length + 1);
                aux_string[aux_length] = '\0';
        }
        else if(first_time[1] == false && rule == RULE5)
        {
                aux_length = strlen(aux_string);
                for(i = aux_length; i > 0; i--)
                        aux_string[i] = aux_string[i - 1];
                aux_string[i] = '1';
                aux_string = realloc(aux_string, aux_length + 1);
                aux_string[aux_length] = '\0';
        }

        return aux_string;
}
```

7. Práctica 6: Autómata de pila que reconoce el lenguaje $\{0^n1^n \mid n \geq 1\}$.

7.1. Descripción

- Los estados del autómata:

- q = estado incial. Nos encontramos en el estado q si hemos visto solo 0's hasta ahora.
- p = hemos visto al menos un 1 y ahora procederemos solo si las entradas son 1's.
- f = estado final; cadena aceptada.

- Los símbolos de la pila:
  - Z = símbolo inicial. Además marca el fondo de la pila, así sabremos cuando hemos contado el mismo número de 0's y de 1's.
  - X = marcador, usado para contar el número de 0's vistos.

- Las transiciones:
  - $\delta(q, 0, Z) = \{(q, XZ)\}$
  - $\delta(q, 0, X) = \{(q, XX)\}$ Estas dos reglas causan que una X sea empujada en la pila por cada 0 leído.
  - $\delta(q, 1, X) = \{(p, \epsilon)\}$ Cuando vemos un 1, vamos al estado p y hacemos un pop a la pila.
  - $\delta(p, 1, X) = \{(p, \epsilon)\}$ Hacemos un pop de una X.
  - $\delta(p, \epsilon, Z) = \{(f, Z)\}$ Acepta en el fondo.

Este programa recibe una cadena desde consola o un archivo (específicando su ruta), también puede generar una cadena aleatoria, y la procesa mostrando en consola y mandando a un archivo usando Descripción Instantánea (ID) e indicándonos si la cadena fue aceptada.

- La ID es una terna $(q, w, \alpha), donde$:
  - q es el estado actual.
  - w es la cadena restante.
  - α representa los contenidos de la pila

Por último señalar que el código del tipo de dato abstracto Pila se encuentra al final de esta sección.

7.2. Ejecución

Al iniciar el programa se nos presenta el menú con el siguiente aspecto:

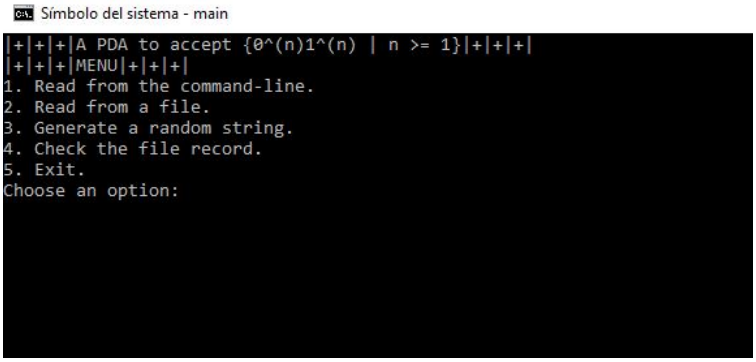


Figura 25: Menú Principal

Si elegimos la opción 1, el programa nos pedirá ingresar la cadena y desplegará la salida posteriormente:

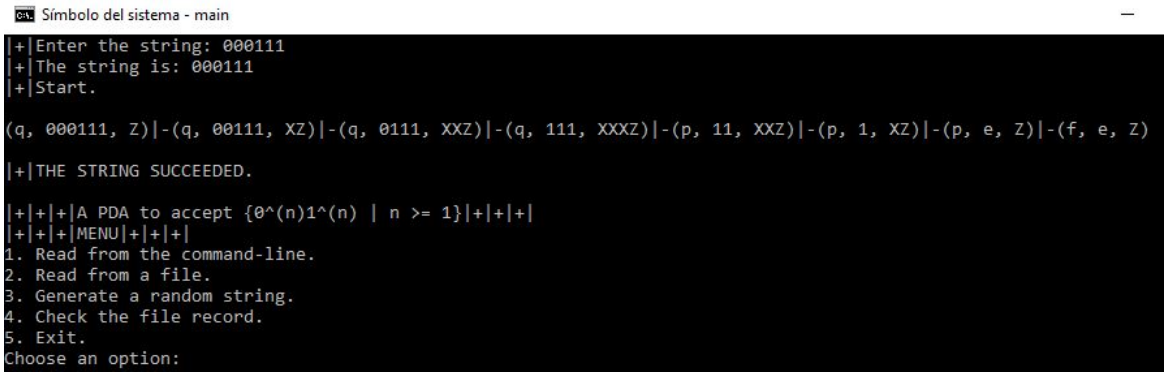


Figura 26: Opción 1

Si elegimos la opción 2, el programa nos pedirá el nombre del archivo desde el cual va a leer la cadena y mostrará la salida correspondiente:

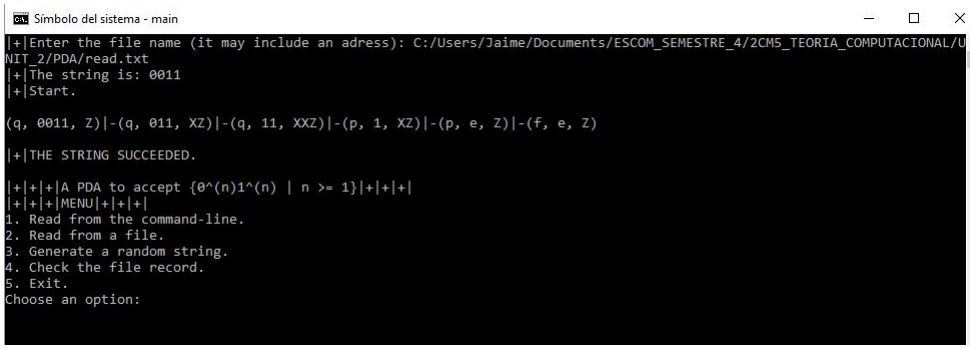


Figura 27: Opción 2

Si elegimos la opción 3, el programa generará una cadena aleatoria que siempre será aceptada por el autómata y que será de longitud menor o igual a 1000:

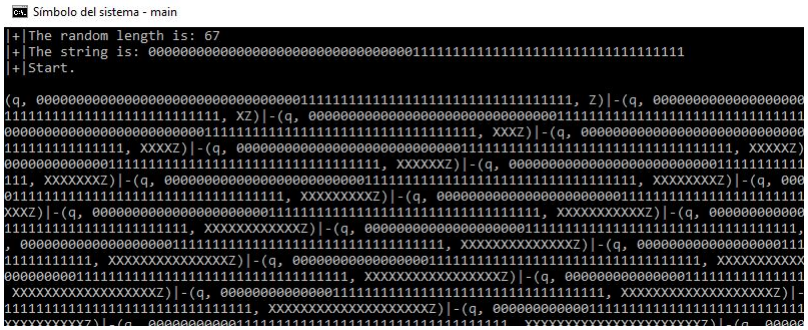


Figura 28: Opción 3

Si elegimos la opción 4, el programa abrirá el archivo en el cual se encuentran todas las descripciones instantáneas, junto con la longitud de la cadena y la cadena:

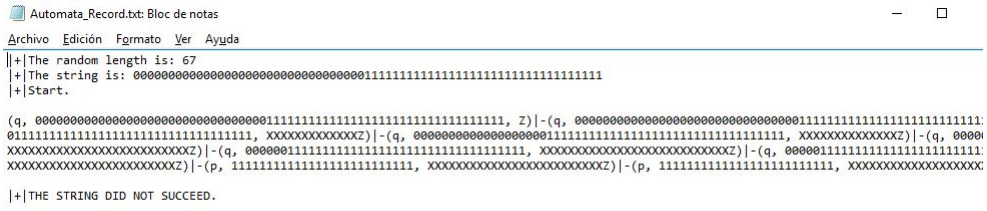


Figura 29: Opción 4

7.3. Código

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include "TADStack.h"

#define STR_LENGTH 10000
#define RANDOM_TOP 1000

long printer = 1;

int main(void)
{
        Stack stack1;
        element e1;
        int option = 0, ch;
        long i = 0, random_length = 0, string_length = 0,
             aux_string_length = 0, j = 0, stack_length = 0;
        char *string, *stack, *aux_string;
        char file_name[FILENAME_MAX];
        FILE *write_fp, *read_fp;

        srand((unsigned) time(NULL));

        for(;;)
        {
                printf("|+|+|+|A PDA to accept {0^(n)1^(n) | n >= 
                    1}|+|+|+|\n");
                printf("|+|+|+|MENU|+|+|+|\n");
                printf("1. Read from the command-line.\n");
                printf("2. Read from a file.\n");
                printf("3. Generate a random string.\n");
                printf("4. Check the file record.\n");
                printf("5. Exit.\n");
                printf("Choose an option: ");
                scanf(" %d", &option);
                switch(option)
                {
                        case 1: case 2: case 3:
                        system("cls");
                        getchar(); //Catches the '\n' entered by
                             the user
                        write_fp = fopen("C:/Users/Jaime/Documents/
                            ESCOM_SEMESTRE_4/2
                            CM5_TEORIA_COMPUTACIONAL/UNIT_2/PDA/
                            Automata_Record.txt", "w");
                        if(option == 1)
                        {
                        string = malloc(STR_LENGTH);
                        printf("|+|Enter the string: ");
                        for(i = 0; (ch = getchar()) != '\n'; i++)
                                string[i] = ch;
                        string[i] = '\0';
                        printf("|+|The string is: %s\n", string);
```

```
fprintf(write_fp, "|+|The string is: %s\n",
     string);
}
else if(option == 2)
{
printf("|+|Enter the file name (it may
    include an adress): ");
scanf("%s", file_name);
if((read_fp = fopen(file_name, "rb")) ==
    NULL)
{
        printf("|+|Can't open: %s",
            file_name);
        break;
}
for(i = 0; (ch = getc(read_fp)) != EOF; i
    ++)
{}
string = malloc(i);
fseek(read_fp, 0, SEEK_SET);
for(i = 0; (ch = getc(read_fp)) != EOF; i
    ++)
        string[i] = ch;
string[i] = '\0';
printf("|+|The string is: %s\n", string);
fprintf(write_fp, "|+|The string is: %s\n",
     string);
}
else if(option == 3)
{
random_length = rand() %RANDOM_TOP + 1;
printf("|+|The random length is: %d\n",
    random_length);
fprintf(write_fp, "|+|The random length is:
    %d\n", random_length);
string = malloc(random_length + 1);
for(i = 0; i < random_length / 2; i++)
        string[i] = '0';
for(i = random_length / 2; i <
    random_length; i++)
        string[i] = '1';
string[i] = '\0';
printf("|+|The string is: %s\n", string);
fprintf(write_fp, "|+|The string is: %s\n",
     string);
}
//AUTOMATA
if(string[0] != '\0' && string[0] == '0')
{
printf("|+|Start.\n\n");
fprintf(write_fp, "|+|Start.\n\n");
Initialize(&stack1);
string_length = strlen(string);
stack = malloc(string_length + 3); //We add
     3 because fuck it
Push(&stack1, (element) {.value = 'Z'});
stack[0] = 'Z';
```

```
stack[1] = '\0';
printf("(q,␣%s,␣%s)|-", string, stack);
fprintf(write_fp, "(q,␣%s,␣%s)|-", string,
    stack);
printer++;
aux_string = malloc(string_length + 1);
strcpy(aux_string, string);
aux_string_length = string_length;
stack_length = 1;

for(i = 0; string[i] != '1'; i++)
{
for(j = 0; j < aux_string_length; j++)
        aux_string[j] = aux_string[j + 1];
aux_string_length--;

for(j = stack_length; j > 0; j--)
        stack[j] = stack[j - 1];
stack[0] = 'X';
stack_length++;
stack[stack_length] = '\0';

printf("(q,␣%s,␣%s)|-", aux_string, stack);
fprintf(write_fp, "(q,␣%s,␣%s)|-",
    aux_string, stack);
printer++;
if(printer == 8)
{
        printf("\n");
        fprintf(write_fp, "\n");
        printer = 1;
}
Push(&stack1, (element) {.value = 'X'});
}
for(; string[i] != '\0'; i++)
{
for(j = 0; j < aux_string_length; j++)
        aux_string[j] = aux_string[j + 1];
aux_string_length--;

for(j = 0; j < stack_length; j++)
        stack[j] = stack[j + 1];
stack_length--;

if(stack[0] == '\0')
{
        Pop(&stack1);
        break;
}
if(aux_string[0] != '\0')
{
printf("(p,␣%s,␣%s)|-", aux_string, stack);
fprintf(write_fp, "(p,␣%s,␣%s)|-",
    aux_string, stack);
printer++;
}
if(printer == 8)
```

```
{
        printf("\n");
        fprintf(write_fp, "\n");
        printer = 1;
}
Pop(&stack1);
}

if(!Empty(&stack1))
{
e1 = Top(&stack1);
if(e1.value == 'Z') //Succeed
{
printf("(p, e, Z)|-(f, e, Z)\n\n");
fprintf(write_fp, "(p, e, Z)|-(f, e, Z)\n\n
    ");
printf("|+|THE STRING SUCCEEDED.\n\n");
fprintf(write_fp, "|+|THE STRING SUCCEEDED
    .\n\n");
}
else //More 0's than 1's
{
printf("(p, e, %s)|-(f, e, %s)\n\n", stack,
     stack);
fprintf(write_fp, "(p, e, %s)|-(f, e, %s)\n
    \n", stack, stack);
printf("|+|THE STRING DID NOT SUCCEED.\n\n"
    );
fprintf(write_fp, "|+|THE STRING DID NOT
    SUCCEED.\n\n");
}
}
else //More 1's than 0's
{
printf("(f, %s, Z)\n\n", aux_string);
fprintf(write_fp, "(f, %s, Z)\n\n",
    aux_string);
printf("|+|THE STRING DID NOT SUCCEED.\n\n"
    );
fprintf(write_fp, "|+|THE STRING DID NOT
    SUCCEED.\n\n");
}
Destroy(&stack1);
}
else
{
printf("|+|Error: The string is empty or it
    begins with 1's.\n\n");
fprintf(write_fp, "|+|Error: The string is
    empty or it begins with 1's.\n\n");
}
fclose(write_fp);
break;
case 4:
system("cls");
system("C:/Users/Jaime/Documents/
    ESCOM SEMESTRE 4/2
```

```
                                CM5_TEORIA_COMPUTACIONAL/UNIT_2/PDA/
                                Automata_Record.txt");
                        break;
                        case 5:
                        system("cls");
                        exit(EXIT_SUCCESS);
                        break;
                        default:
                        system("cls");
                        printf("Choose a correct option.\n");
                        Sleep(2500);
                        break;
                }
        }
        return 0;
}
```

7.3.1. Código del TADStack.h

```
#include <stdbool.h>

typedef struct element
{
    char value;
        struct element *down;
}element;


typedef element *Stack;

//DECLARACIN DE FUNCIONES
void Initialize(Stack *stack);                  //Inicializar pila
    (Iniciar una pila para su uso)
void Push(Stack *stack, element e);             //Empilar (
    Introducir un elemento a la pila)
element Pop(Stack *stack);                              //
    Desempilar (Extraer un elemento de la pila)
bool Empty(Stack *stack);                               //Vacia (
    Preguntar si la pila esta vacia)
element Top(Stack *stack);                              //Tope (
    Obtener el "elemento" del tope de la pila si extraerlo de la
    pila)
int ValueTop(Stack *stack);                             //Tama o
    de la pila (Obtener el n mero de elementos en la pila)
void Destroy(Stack *stack);                             //Elimina
    pila (Borra a todos los elementos y a la pila de memoria)
```

7.3.2. Código del TADStack.c

```
#include <stdlib.h>
#include "TADStack.h"

void Initialize(Stack *stack)
```

```
{
        *stack = NULL;
        return;
}

void Push(Stack *stack, element e)
{
        element *aux;

        aux = malloc(sizeof(element));
        *aux = e;
        aux->down = *stack;
        *stack = aux;

        return;
}

element Pop(Stack *stack)
{
        element e, *aux;

        e = **stack;
        aux = *stack;
        *stack = (*stack)->down;
        free(aux);

        return e;
}

bool Empty(Stack *stack)
{
        if(*stack == NULL)
                return true;
        return false;
}

element Top(Stack *stack)
{
    return **stack;
}

int ValueTop(Stack *stack)
{
        element *aux;
        int stack_size = 0;

        aux = *stack;
        if(aux != NULL)
        {
                stack_size++;
                while(aux->down != NULL)
                {
                        stack_size++;
                        aux = aux->down;
                }
        }
        return stack_size;
```

```
}

void Destroy(Stack *stack)
{
        element *aux;

        if(!Empty(stack))
        {
                while(*stack != NULL)
                {
                        aux = (*stack)->down;
                        free(*stack);
                        *stack = aux;
                }
        }
        return;
}
```