EIE

Escuela de
Ingeniería Eléctrica

UNIVERSIDAD DE COSTA RICA

UNIVERSIDAD DE COSTA RICA

FACULTAD DE INGENIERIA

ESCUELA DE INGENIERIA ELÉCTRICA

PROGRAMACIÓN BAJO PLATAFORMAS ABIERTAS

LABORATORIO 7:
RECURSIVIDAD

ESTUDIANTE:
JESÚS ZÚÑIGA MÉNDEZ

PROFESOR:
RICARDO ROMÁN BRENES; M. SC.

I CICLO 2019

Índice

Índice de figuras

1. Introducción

Para este laboratorio el objetivo fue crear un programa que pudiera recorrer una matriz a manera de laberinto de forma recursiva, a continuación se adjunta el código escrito en C con el que se soluciono el problema, en el código se encuentra la documentación necesaria para comprender que lógica que siguen las funciones presentes en el programa. Es importante aclarar que hay código escrito que no se usa ya que se implemento para crear laberintos aleatorios, sin embargo por razones de tiempo no se pudieron crear.

2. Solucion

La función principal buscarQueso es la que se encarga de encontrar el camino del laberinto, para poder construirla primero se necesito definir las posiciones a las que se puede mover, es decir las que no contienen el símbolo de número, después de definir estas posiciones simplemente es de ir moviéndose casilla por casilla primero arriba, después a la derecha, luego abajo y por ultimo a la izquierda, si no se puede mover a un campo libre se empieza a mover por donde ya ha transitado, esto así hasta encontrar el queso, en esta función se implemento otro símbolo el "*.[el] cual fue necesario para lograr recorrer laberintos mas difíciles como el laberinto2, el cual contiene bucles y caminos anchos, las demás funciones usadas en este laboratorio se han usado en otros laboratorios anteriores por lo que no se explican.

3. includes.h

```
#ifndef INCLUDES_H
#define INCLUDES_H
    typedef struct _posicion posicion;
    typedef struct _lista lista;

    typedef
    struct _matriz
    {
        char** tablero;
        int filas;
        int columnas;
        int semilla;
        int posicion[2];
        lista* pasos;
    }
    matriz;

    #include <stdio.h>
    #include <stdlib.h>
    #include <time.h>
    #include "./Laberinto.h"
    #include "./Recorrer.h"
    #include "./lista.h"
    #include "./posicion.h"
#endif
```

4. posicion.h

```
#ifndef POSICION1_H
#define POSICION1_H

#include "includes.h"
    typedef
        struct _posicion
        {
            int d;
            struct _posicion* siguiente;
        }
    posicion;
    posicion* Siguiente(posicion*);
    posicion* Anterior(posicion*, lista*);
    posicion* CrearPosision(int);
    void EliminarPosicion(posicion*);

#endif
```

5. lista.h

```
#ifndef LISTA_H
#define LISTA_H
#include "includes.h"

    typedef
        struct _lista
        {
            unsigned int items;
            posicion* primero;
        }
    lista;

    lista* CrearLista();
    void ImprimirLista(lista*);
    void EliminarLista(lista*);
    void AgregarElemento(lista*, int);
#endif
```

6. Laberinto.h

```
#ifndef LABERINTO_H
#define LABERINTO_H
    char** CrearTablero (int, int);
    //int NumeroRandom(int);
    void Esperar(float);
    void LlenarGatos(matriz*);
    void ImprimirMatriz(matriz*);
    void GuardarLaberinto(matriz *);
    void HacerCamino(matriz *);
    void EscribirCaracter(matriz*, char);
    int MePuedoMover(matriz*, int);
    //int NumeroRandomSemilla(int);
#endif
```

7. Recorrer.h

```
#ifndef RECORRER_H
#define RECORRER_H
    void LLenarLaberinto(matriz* , char**);
    int VerificarLaberinto(matriz* , char** );
    void LlenarTablero(matriz*, char**);
    void UbicarRaton(matriz* );
    void buscarQueso(matriz*,int, int);
#endif
```

8. posicion.c

```
/**
 * @file posicion.c
 * @author Jesus Zuñiga Mendez
 * @brief Archivo que contiene las funciones para manipular las posiciones de una ñista
 */
#include "../include/includes.h"
/**
 * @brief Funcion que devuelve la posicion siguiente de una lista
 * @param p posicion inicial
 */
posicion* Siguiente(posicion* p){
    posicion* Np = p->siguiente;
    return Np;
}
/**
 * @brief Funcion que devuelve la posicion anterior de una lista
 * @param p posicion inicial
 * @param ls lista en la que se va a buscar
 */
posicion* Anterior(posicion* p, lista* ls){
    posicion* anterior=ls->primero;
    posicion* actual = ls-> primero;
    if (actual != 0x0){
        for (int i=1; i<= ls->items; i++){
            if (actual == p){
                i=ls->items;
            }else
            {
                anterior = actual;
                actual = actual->siguiente;
            }

        }
    }
    return anterior;
}
/**
 * @brief Funcion que crea una posicion vacia
 * @param dato Es el dato que se ve a agregar
 */
posicion* CrearPosision(int dato){
    posicion* p = (posicion*) malloc(sizeof(posicion));
    p->d= dato;
    return p;
}
/**
 * @brief Funcion que elimina una poicion
 * @param p es el posicion que se ve a eliminar
 */
void EliminarPosicion(posicion* p){
    free(p);
}
```

9. lista.c

```
/**
 * @file lista.c
 * @author Jesus Zuñiga Mendez
 * @brief Archivo que contiene las funciones para manipular listas
 */
#include "../include/includes.h"


/**
 * @brief Funcion que crea una lista vacia
 * @return ls es la lista que se acaba de crear;
 */

lista* CrearLista(){
    lista* ls = (lista*) malloc (sizeof(lista));
    ls->items = 0;
    ls->primero = 0x0;
    return ls;
}


/**
 * @brief Funcion que elimina una lista
 * @param ls lista con la que se va a trabajar
 */

void EliminarLista(lista* ls){
    posicion* q;
    posicion* p = ls->primero;
    for (int i = 0; i < ls->items; i++){
        q = Siguiente(p);
        EliminarPosicion(p);
        p=q;
    }
    free(ls);
}

/**
 * @brief Funcion que agrega un elemento a la lista
 * @param ls lista con la que se va a trabajar
 * @param elemento elemento que se va a agregar
 */
void AgregarElemento(lista* ls, int elemnto){
    posicion* nuevaP= CrearPosision(elemnto);
    posicion* p = ls -> primero;
    if (p == 0x0){
        ls -> primero = nuevaP;
        nuevaP ->siguiente = 0x0;
        ls ->items++;
    }else{
        for (int i = 1; i< ls->items; i++){
            p = Siguiente(p);
```

```
        }
        p->siguiente= nuevaP;
        nuevaP ->siguiente = 0x0;
        ls->items++;
    }


}

/**
 * @brief Funcion que imprime una lista
 * @param ls lista que se va a imprimir
 */
void ImprimirLista(lista* ls){
    posicion* p = ls->primero;
    for (int i = 0; i < ls->items-1; i++)
    {
        p = Siguiente(p);
    }

    for (int i = 0; i < ls->items; i++){
        switch (p->d)
        {
        case 1:
            printf("Arriba\n");
            break;
        case 2:
            printf("Derecha\n");
            break;
        case 3:
            printf("Abajo\n");
            break;
        case 4:
            printf("Izquierda\n");
            break;
        case 5:
            printf("Paso atras hacia Arriba\n");
            break;
        case 6:
            printf("Paso atras hacia la Derecha\n");
            break;
        case 7:
            printf("Paso atras hacia Abajo\n");
            break;
        case 8:
            printf("Paso atras hacia la Izquierda\n");
            break;
        case 9:
            printf("Paso atras hacia Arriba\n");
            break;
        case 10:
            printf("Paso atras hacia la Derecha\n");
            break;
        case 11:
```

```
            printf("Paso atras hacia Abajo\n");
            break;
        case 12:
            printf("Paso atras hacia la Izquierda\n");
            break;
        default:
            break;
        }
        p = Anterior(p, ls);
    }
    printf("\n");
}
```

10. Laberinto.c

```
/**
 * @file Laberinto.c
 * @author Jesus Zuñiga Mendez
 * @brief Archivo que contiene las funciones para Crear un laberinto
 */

#include "../include/includes.h"

//controlan los colores de impresion
#define ANSI_COLOR_RED     "\x1b[31m"
#define ANSI_COLOR_GREEN   "\x1b[32m"
#define ANSI_COLOR_YELLOW  "\x1b[33m"
#define ANSI_COLOR_BLUE    "\x1b[34m"
#define ANSI_COLOR_MAGENTA "\x1b[35m"
#define ANSI_COLOR_CYAN    "\x1b[36m"
#define ANSI_COLOR_RESET   "\x1b[0m"
#define ANSI_COLOR_BOLD_YELLOW   "\x1b[01;33m"
#define ANSI_COLOR_BOLD_GREEN   "\x1b[01;32m"



/**
 * @brief Funcion que crea una matriz del tamaño deseado por el usuario
 * @param filas es el tamaño de filas de la matriz
 * @param columnas es el tamaño de columnas de la matriz
 * @return devuelve un puntero con la matriz creada
 */
char** CrearTablero (int filas, int columnas){
    //creamos el puntero con "forma de matriz" y le asignamos el tamaño a las filas
    char** matriz = (char**) malloc(filas * sizeof(char*));
    //este ciclo toma el puntero lo "recorre" por filas y le asigna el tamaño de las column
    for (int i = 0; i < filas; i++){
        *(matriz+i) = (char*) malloc(columnas * sizeof(char));
    }
    return matriz;
}



/**
 * @brief Funcion que detiene la ejecucion cierta cantidad de segundos
 * @param tiempo el el tiempo en espera en segundos
 * codigo recuperado de https://stackoverflow.com/questions/7684359/how-to-use-nanosleep-in
 */
void Esperar(float segundos){
    struct timespec tim, tim2;
    tim.tv_sec = segundos;
    if(nanosleep(&tim , &tim2) < 0 )
    {
        Esperar(segundos);
    }
```

```
}

/**
 * @brief Funcion que llena una metriz de simbolos de # exeptuando las filas y columnas ext
 * @param recibe un puntero a la estructura matriz
 */
void LlenarGatos(matriz* laberinto){
    char gato = '#';
    char neutro = 'N';
    for(int f = 0; f < laberinto->filas; f++){
        for(int c = 0; c < laberinto->columnas; c++){
            if ((f == 0) || (c==0) || (f==(laberinto->filas-1)) || (c==(laberinto->columnas
                //*(*(laberinto->tablero+f)+c) = neutro;
                *(*(laberinto->tablero+f)+c) = gato;
            }else
            {
                *(*(laberinto->tablero+f)+c) = gato;
            }

        }
    }
}


/**
 * @brief Funcion que permite imprimir una matriz
 * @param recibe un puntero a la matriz
 */
void ImprimirMatriz(matriz* laberinto){
    char simbolo = ' ';
    printf("\n\n\n");
    for(int f = 0; f < laberinto->filas; f++){
        for(int c = 0; c < laberinto->columnas; c++){
            simbolo = *(*(laberinto->tablero+f)+c);
            if (simbolo == '#'){
                printf(ANSI_COLOR_BOLD_GREEN"");
            }
            if (simbolo == ' '){
                printf(ANSI_COLOR_RESET"");
            }
            if (simbolo == '.'){
                printf(ANSI_COLOR_YELLOW"");
            }
            if (simbolo == '*'){
                simbolo = '.';
                printf(ANSI_COLOR_YELLOW"");
            }
            if (simbolo == 'Q'){
                printf(ANSI_COLOR_BOLD_YELLOW"");
            }
            if (simbolo == 'r'){
                printf(ANSI_COLOR_RESET"");
            }
            printf("%c"ANSI_COLOR_RESET,simbolo);
```

```
        }
    printf("\n");
    }
}


/**
 * @brief Funcion que escribe un caracter en una coordenada indicada
 * @param laberinto es la estructura tipo matriz que contiene la matriz de caracteres
 * @param caracter es el caracter que se quiere escribir
*/
void EscribirCaracter(matriz* laberinto, char caracter){
    int fila = laberinto->posicion[0];
    int columna = laberinto->posicion[1];
    *(*(laberinto->tablero+fila)+columna) = caracter;
}


/**
 * @brief Funcion que guarda una matriz en un archivo
 * @param laberinto es la estructura tipo matriz que contiene la matriz de caracteres
*/
void GuardarLaberinto(matriz * laberinto)
{
    int s = 0;
    char nombre[128];
    char simbolo = ' ';
    FILE* f_out;
    printf("Digite el nombre del laberinto nuevo.\n");
    s = scanf("%s",nombre);
    f_out = fopen(nombre, "w");
    for(int f = 0; f < laberinto->filas; f++){
        for(int c = 0; c < laberinto->columnas; c++){
            simbolo = *(*(laberinto->tablero+f)+c);
            fprintf(f_out, "%c",simbolo);
        }
    fprintf(f_out, "\n");
    }
    fclose(f_out);
}


/**
 * @brief Funcion que crea el camino principal es decir el camino ganador aleatoriamente
 * @param laberinto es la estructura tipo matriz que contiene la matriz de caracteres
*/
void HacerCamino(matriz* laberinto){
    //primero dibujamos una coordenada aleatoria que es desde donde se empieza a dibjar el
    int f =0;
    int c = 1;
    //primero nos ubicamos en una coordenada aleatoria para ubicar el caracter
    laberinto->posicion[f] = (rand() + laberinto->semilla) % (laberinto->filas);
    laberinto->posicion[c] = (rand() + laberinto->semilla) % (laberinto->columnas);
    //verificamos que no sea un numero negativo y si lo es lo hacemos positivo
```

```
    if (laberinto->posicion[f]<0){
        laberinto->posicion[f]=laberinto->posicion[f]*-1;
    }
    if (laberinto->posicion[c]<0){
        laberinto->posicion[c]=laberinto->posicion[c]*-1;
    }
    //verificamos que el numero se encuentre entre los limites, un limite es la segunda col
    //si se pasa de los limites lo colocamos en el centro del laberinto
    if ((laberinto->posicion[f]<1) || (laberinto->posicion[f]>(laberinto->filas-2))){
        laberinto->posicion[f]=(int)laberinto->filas/2;
    }
    if ((laberinto->posicion[c]<1) || (laberinto->posicion[c]>(laberinto->columnas-2))){
        laberinto->posicion[c]=(int)laberinto->columnas/2;
    }
    EscribirCaracter(laberinto, ' ');

    //despues de esto vamos a empezar a dibujar el camino a partir de esta posicion
    int parar = 0;
    int lado = 0;
    int mover = 0;
    //pedimos una direccion para dibujar
    //despues vamoa a empezar a dibujar el camino a partir de la entrada
    while(parar != 1){
        laberinto->semilla = ((rand() + laberinto->semilla) % 4);
        lado = laberinto->semilla;
        mover = MePuedoMover(laberinto, lado);
        if (lado == 0){
            //vamos a movernos a la izquirda

        }else if (lado == 1){
            //vamos a movernoas a la derecha
        }else if (lado == 2){
            //vamos a movernoas a arriba
        }else if (lado == 3){
            //vamos a movernoas abajo
        }
    }
}

/**
 * @brief Funcion que verifica que se cumplan los requisitos para moverse
 * @param laberinto es la estructura tipo matriz que contiene la matriz de caracteres
 * Qparam lado es la direccion, 0 = izquirda, 1= derecha, 2= arriba, 3= abajo
*/
int MePuedoMover(matriz* laberinto, int lado){
    int respuesta = 0;
    /*if (lado == 0){
        //vamos a movernos a la izquirda
    }else if (lado == 1){
        //vamos a movernoas a la derecha
    }else if (lado == 2){
        //vamos a movernoas arriba
    }else if (lado == 3){
        //vamos a movernoas abajo
```

```
    }*/
    return respuesta;
}
```

11. Recorrer.c

```
/**
 * @file Recorrer.c
 * @author Jesus Zuñiga Mendez
 * @brief Archivo que contiene las funciones para Recorrer un laberinto
 */
#include "../include/includes.h"


/**
 * @brief Funcion que llena un laberinto con los datos del archivo sellecionado por el usua
 * @param laberinto es la estructura que contiene todos los datos
 * @param archivo es la direccion donde esta el archivo
 */
void LLenarLaberinto(matriz* laberinto, char** Archivo){
    int verificar = VerificarLaberinto(laberinto, Archivo);
    if (verificar == 1){
        laberinto->tablero = CrearTablero(laberinto->filas,laberinto->columnas);
        LlenarTablero(laberinto,Archivo);
        printf("Laberinto cargado con exito\n\n\n");
        UbicarRaton(laberinto);
        //printf("Las coordenadas del raton son fila %d y columna %d\n\n\n",laberinto->posi
        laberinto->pasos = CrearLista();
        laberinto->pasos->items = 0;
        laberinto->pasos->primero = 0x0;
        buscarQueso(laberinto, laberinto->posicion[0], laberinto->posicion[1]);
        printf("\n\n\n");
    }else
    {
        printf("Laberinto DAÑADO\n");
    }


}

/**
 * @brief Funcion que ubica la posicion inicial del raton
 * @param laberinto es la estructura que contiene todos los datoss
 */
void UbicarRaton(matriz* laberinto){
    char simbolo = 'r';
    for(int f = 0; f < laberinto->filas; f++){
        for(int c = 0; c < laberinto->columnas; c++){
            if (simbolo == *(*(laberinto->tablero+f)+c)){
                laberinto->posicion[0] = f;
                laberinto->posicion[1] = c;
            }
        }
    }
}

/**
 * @brief Funcion que ubica la posicion inicial del raton
```

```
 * @param laberinto es la estructura que contiene todos los datoss
 */
void buscarQueso(matriz* laberinto,int fila, int columna){
    ImprimirMatriz(laberinto);
    Esperar(1);
    int limiteFilas = laberinto->filas-1;
    int limiteColumnas = laberinto->columnas-1;
    char izquierda='V';
    char derecha='V';
    char arriba='V';
    char abajo='V';
    char punto = '.';
    char raton = 'r';
    int paso=0;
    int f=fila;
    int c=columna;
    int posicionFAnterior = 0;
    int posicionCAnterior = 0;
    f=f-1;
    if (f>=0){
        arriba = *(*(laberinto->tablero+f)+c);
    }
    f=fila;
    f=f+1;
    if (f<=limiteFilas){
        abajo = *(*(laberinto->tablero+f)+c);
    }
    f=fila;
    c=c-1;
    if (c>=0){
        izquierda = *(*(laberinto->tablero+f)+c);
    }
    c=columna;
    c=c+1;
    if (c<=limiteColumnas){
        derecha = *(*(laberinto->tablero+f)+c);
    }
    c=columna;
    //caso basico cuando el queso esta al lado
    if ((izquierda == 'Q') || (derecha=='Q') || (arriba=='Q') || (abajo=='Q')){
        printf("\n\n");
        printf("Encontre el queso\n");
    }else{
        *(*(laberinto->tablero+f)+c) = punto;
        posicionFAnterior = laberinto->posicion[0];
        posicionCAnterior = laberinto->posicion[1];
        laberinto->posicion[0] = f;
        laberinto->posicion[1] = c;
        if (arriba == ' '){
            paso=1;
            f=f-1;
        }else if (derecha == ' '){
            paso=2;
            c=c+1;
```

```
        }else if (abajo == ' '){
            paso=3;
            f=f+1;
        }else if (izquierda == ' '){
            paso=4;
            c=c-1;
        }//se uso un caracter asterisco para poder recorrer laberintos mas complicaados con
         //el cual tiene un blucle y caminos con mas de un caracter de espacio
        else if ((arriba == '.') && (posicionFAnterior != (f-1))){
            paso=5;
            *(*(laberinto->tablero+f)+c) = '*';
            f=f-1;
        }else if ((derecha == '.')&& (posicionCAnterior != (c+1))){
            paso=6;
            *(*(laberinto->tablero+f)+c) = '*';
            c=c+1;
        }else if ((abajo == '.') && (posicionFAnterior != (f+1))){
            paso=7;
            *(*(laberinto->tablero+f)+c) = '*';
            f=f+1;
        }else if ((izquierda == '.') && (posicionCAnterior != (c-1))){
            paso=8;
            *(*(laberinto->tablero+f)+c) = '*';
            c=c-1;
        }else if ((arriba == '*') && (posicionFAnterior != (f-1))){
            paso=9;
            *(*(laberinto->tablero+f)+c) = '*';
            f=f-1;
        }else if ((derecha == '*')&& (posicionCAnterior != (c+1))){
            paso=10;
            *(*(laberinto->tablero+f)+c) = '*';
            c=c+1;
        }else if ((abajo == '*') && (posicionFAnterior != (f+1))){
            paso=11;
            *(*(laberinto->tablero+f)+c) = '*';
            f=f+1;
        }else if ((izquierda == '*') && (posicionCAnterior != (c-1))){
            paso=12;
            *(*(laberinto->tablero+f)+c) = '*';
            c=c-1;
        }
        *(*(laberinto->tablero+f)+c) = raton;
        buscarQueso(laberinto,f,c);
    }
    AgregarElemento(laberinto->pasos,paso);
}



/**
 * @brief Funcion que llena el tablero con los datos del laberinto
 * @param laberinto es la estructura que contiene todos los datos
 * @param Archivo es el arreglo que contien la ruta del archivo
 */
```

```
void LlenarTablero(matriz* laberinto, char** Archivo){
    //abrimos el archivo
    char caracter;
    char salto = '\n';
    char caracteresValidos[4]= {'#',' ','r','Q'};
    FILE* archivo;
    archivo = fopen(Archivo[1], "r");
    int f=0;
    int c=0;
    caracter = fgetc(archivo);
    while (caracter != -1){
        if ((caracter == caracteresValidos[0]) || (caracter == caracteresValidos[1]) || (ca
            *(*(laberinto->tablero+f)+c) = caracter;
            c++;
            caracter = fgetc(archivo);
        }else if(caracter == salto){
            f++;
            c=0;
            caracter = fgetc(archivo);
        }else
        {
            caracter = fgetc(archivo);
        }

    }
    fclose(archivo);

}

/**
 * @brief Funcion que revisa que el laberinto este escrito de forma correcta, ademas modifi
 * @param laberinto es la estructura que contiene todos los datos
 * @param Archivo es el arreglo que contien la ruta del archivo
 * @return rspuesta es un indicador si el laberinto es correcto o no
 */
int VerificarLaberinto(matriz* laberinto, char** Archivo){
    int respuesta = 0;
    //abrimos el archivo
    FILE* archivo;
    archivo = fopen(Archivo[1], "r");
    int filas = 1;
    int columnas = -1;
    char salto = '\n';
    char caracter;
    int contador = 0;
    int comprobarFormato = 1;
    int comprobarDimensiones = 0;
    int comprobarDimensiondesDos = 0;
    char caracteresValidos[4]= {'#',' ','r','Q'};
    //vamos a recorrer un vez el laberinto para medir las dimensiones y para verificar que
    while (contador < 1){
        comprobarFormato = 1;
        caracter = fgetc(archivo);
        //verificamos que el caracter sea valido
```

```
        for (int i=0; i < 4; i++){
            if (caracter == caracteresValidos[i]){
                comprobarFormato = 0;
            }
            if (caracter == salto){
                comprobarFormato = 0;
            }
            if (caracter == 13){
                comprobarFormato = 0;
            }
            if (caracter == 10){
                comprobarFormato = 0;
            }

        }
        //si no es valido nos salimos del ciclo
        contador = comprobarFormato;
        if (caracter == salto){
            filas++;
            columnas = -1;
        }else{
            columnas++;
        }
        if(caracter == -1)
        {
            contador = 1;
        }
        if((caracter != -1) && (caracter != 13) && (caracter != 10)){
            comprobarDimensiones++;
        }
    }
    comprobarDimensiondesDos = filas * columnas;
    if (comprobarDimensiones == comprobarDimensiondesDos){
        respuesta = 1;
        laberinto->filas = filas;
        laberinto->columnas = columnas;
    }else{
        respuesta = 0;
    }
    fclose(archivo);
    return respuesta;
}
```

12. main.c

```
/**
 * @file main.c
 * @author Jesus Zuñiga Mendez
 * @brief Archivo pricipal, laboratorio que trata sobre listas enlazadas
 * @version 1.0
 * @date 20 de junio de 2019
 * @copyright Copyleft (l) 2019
 */
#include "./include/includes.h"

/**
 * @brief Función que imprime una bienvenida para el usuario
 */
void Bienvenida(){
    printf("  ____   _                                _      _          _  _  _ \n |  _ \'
    printf("\n");
    printf("\n");
}
/**
 * @brief Funcion que imprime el menu
 * @return seleccion Es la opcion escogida por el usuario
 */
int Menu(){
    printf("Digite el numero de la opcion que quiere realizar.\n\n\n");
    printf("1. Crear un laberinto aleatorio nuevo.\t\n");
    printf("2. Seleccionar laberinto.\t\n");
    printf("3. Recorrer laberinto.\t\n");
    printf("0. SALIR.\t\n\n\n");
    int seleccion = 0;
    int s = scanf("%d",&seleccion);
    return seleccion;
}
/**
 * @brief Funcion que se encarga de ejecurar los comandos para crear laberinto nuevo
 */
void OpcionUno(){
    int s = 0;
    matriz *laberinto = (matriz*) malloc (sizeof(matriz));
    laberinto->posicion[0] = 0;
    laberinto->posicion[1] = 0;
//   printf("Digite el tamaño maximo de FILAS\n");
//    s=scanf("%d",&laberinto->filas);
    laberinto->filas=21;
//    laberinto->filas = NumeroRandom(laberinto->filas);
//    printf("Digite el tamaño maximo de COlLUMNAS\n");
//    s=scanf("%d",&laberinto->columnas);
    laberinto->columnas=21;
//    printf("Digite la seed del camino\n");
//    s=scanf("%d",&laberinto->semilla);
    laberinto->semilla = 1215611;
//    laberinto->columnas = NumeroRandom(laberinto->columnas);
    laberinto->tablero = CrearTablero(laberinto->filas,laberinto->columnas);
```

```
    LlenarGatos(laberinto);
    HacerCamino(laberinto);
    ImprimirMatriz(laberinto);
    //GuardarLaberinto(laberinto);
    //LimpiarTablero(nuevoTablero);
}

/**
 * @brief Funcion que se encarga de ejecurar los comandos para recorrer un laberinto
 * @param Archivo es la ubicacion del archivo
 */
void OpcionTres(char** Archivo){
    matriz *laberinto = (matriz*) malloc (sizeof(matriz));
    LLenarLaberinto(laberinto, Archivo);
    printf("Segui los siguientes pasos\n\n\n");
    ImprimirLista(laberinto->pasos);
}


/**
 * @brief Funcion Principal, contiene el codigo que ejecuta las demas acciones
 */
int main(int argc, char** argv){
    Bienvenida();
    int opcion = 0;
    do{
        opcion = Menu();
        switch (opcion)
        {
        case 1:
            OpcionUno();
        case 2:
            break;
        case 3:
            OpcionTres(argv);
            break;
        case 0:
            printf("cero\n");
            break;
        default:
            printf("seleccione un numero correcto\n");
            break;
        }
    }while (opcion !=0);
    return 0;
}
```

13. Conclusión

Se concluye que las funciones recursivas permiten crear un proceso que logra resolver ciertos problemas con una cantidad de código un poco mas reducida, ademas de lograr completar un problema de una forma mas concisa y clara.