

Estructura de Datos y Análisis de Algoritmos

Experiencia 1: Procesamiento de imágenes

Andrés Felipe Muñoz Bravo
19.646.487-5

Profesor:
Mario Inostroza

Ayudante:
Javiera Torres M.

Santiago - Chile
2016

TABLA DE CONTENIDOS

Tabla de Contenidos.....	I
Índice de Figuras	II
Índice de Tablas	II
CAPÍTULO 1. Introducción	1
CAPÍTULO 2. Descripción de la solución	2
2.1 Estructuras	2
2.2 Funciones	3
2.2.1 Cargar imagen principal:	3
2.2.2 Contar las imágenes a buscar:	5
2.2.3 Cargar las imágenes a buscar:	6
2.2.4 Buscar las imágenes:	7
2.2.5 Liberar espacio utilizado	9
CAPÍTULO 3. Análisis de los resultados	10
CAPÍTULO 4. Conclusión.....	11
CAPÍTULO 5. Referencias	11
CAPÍTULO 6. Manual de usuario	12
2.3 Introducción	12
2.4 Como compilar y ejecutar	13
2.4.1 Compilar y ejecutar en Windows:	13
2.4.2 Compilar y ejecutar en Linux:.....	15
2.5 Funcionalidades del programa	17
2.6 Posibles errores	17

ÍNDICE DE FIGURAS

<i>Figura 1: Estructuras.....</i>	<i>pag 3</i>
<i>Figura 2: Función main.</i>	<i>pag 4</i>
<i>Figura 3: Función cargarImagenPrincipal..</i>	<i>pag 5</i>
<i>Figura 4: Función incializarImagen.</i>	<i>pag 5</i>
<i>Figura 5: Función cargarImagen..</i>	<i>pag 6</i>
<i>Figura 6: Función contarImagenes.</i>	<i>pag 6</i>
<i>Figura 7: Función cargarImagenesBuscar..</i>	<i>pag 7</i>
<i>Figura 8: Función BusquedaTotal.</i>	<i>pag 8</i>
<i>Figura 9: Función buscarImagen.</i>	<i>pag 9</i>
<i>Figura 10: Función rotarImagen.</i>	<i>pag 9</i>
<i>Figura 11: Función liberarImagen.</i>	<i>pag 10</i>
<i>Figura 12: Función liberarImagenesArreglo..</i>	<i>pag 10</i>

ÍNDICE DE TABLAS

<i>Tabla 1: Funciones y orden de algoritmo</i>	<i>pag 11</i>
--	---------------

CAPÍTULO 1. INTRODUCCIÓN

Se ha pedido realizar un programa que verifique si se encuentran algunas imágenes dentro de una imagen principal. Las imágenes a buscar se encuentran en un archivo de texto llamado “imagesBuscar.txt” las cuales puede estar rotadas en 90, 180, 270 grados y la imagen principal en un archivo llamado “imagenPrincipal.txt”. El programa analiza estos archivos de texto y entrega los resultados en un archivo llamado “resultado.txt”, en el cual se especifica si fueron encontradas las imágenes como también las que no fueron encontradas.

La aplicación es programada en el lenguaje de programación C, además se utiliza el paradigma de programación imperativo procedural.

CAPÍTULO 2. DESCRIPCIÓN DE LA SOLUCIÓN

Para el desarrollo de este programa es utilizada la herramienta sublime text 3, el cual es un editor de texto. La principal ventaja de este editor es que utiliza colores para las diferentes palabras reservadas del lenguaje de programación C, de esta forma ayuda al programador revisar la sintaxis programada. A continuación se explicara detalladamente las estructuras y las principales funciones del programa realizado.

2.1 ESTRUCTURAS

Para el programa se utilizan dos estructuras (pixel_t e imagen_t) que se muestran a continuación:

```
struct pixel{
    int r;
    int g;
    int b;
};
typedef struct pixel pixel_t;

struct imagen{
    int fila;
    int column;
    pixel_t **píxeles;
};
typedef struct imagen imagen_t;
```

Figura 1: Estructuras

Dentro de la estructura de pixel_t se encuentran definido 3 enteros que obtienen un valor entre 0 y 255, donde “r” representa la cantidad de rojo, “g” la cantidad de verde y “b” la cantidad de azul que existe en ese pixel.

Por otro lado se encuentra la estructura imagen_t la cual en palabras simples es una matriz que contiene píxeles, en representación de una imagen. Donde “fila” es el lardo de la imagen, “columna” es el ancho de la imagen, “**píxeles” representa un arreglo bidimensional (la matriz), que contiene píxeles (estructura anterior).

2.2 FUNCIONES

A continuación se muestran los pasos y las funciones principales utilizadas por el programa. Cabe recalcar que dentro de algunas funciones subyacen otras funciones.

```
int main()
{
    // CARGAR LAS IMAGENES:

    imagen_t* imagenPrincipal;
    imagenPrincipal=cargarImagenPrincipal(imagenPrincipal);

    imagen_t** arregloImagenes;
    int cantidadImagenes=contarImagenes();
    arregloImagenes=cargarImagenesBuscar(arregloImagenes,cantidadImagenes);
    //TERMINO DE CARGAR LAS IMAGENES.

    //INICIO ALGORITMO DE BUSQUEDA DE LAS IMAGENES:
    BusquedaTotal(cantidadImagenes,arregloImagenes,imagenPrincipal);
    //TERMINO DE ALGORITMO DE BUSQUEDA DE LAS IMAGENES:

    //LIBERAR EL ESPACIO UTILIZADO POR LAS IMAGENES CARGADAS:
    liberarImagen(imagenPrincipal);
    liberarImagenesArreglo(arregloImagenes,cantidadImagenes);
    //TERMINO DE LA LIBERACION DE LAS IMAGENES.
    return 0;
}
```

Figura 2: Función main.

2.2.1 Cargar imagen principal:

imagen_t cargarImagenPrincipal(imagen_t* imagen):* Esta función es utilizada para introducir todos los datos de los píxeles leídos desde el archivo de texto “imagenPrincipal.txt” en la imagen principal, luego retorna un puntero de esta “imagen” con todos los datos.

Esta función necesita de donde funciones, por lo cual esta función se encarga de recolectar y entregar los datos necesarios para ellas leyendo línea a línea un archivo de texto. Más adelante se explican estas funciones.

```

imagen_t* cargarImagenPrincipal(imagen_t* imagen){

    FILE* archivo;
    archivo = fopen("Entrada/imagenPrincipal.txt","r");
    char cadena[LARGOLINE];
    fgets(cadena,LARGOLINE,archivo);
    int fila,columna;
    if (buscarLinea(cadena)){
        buscarDimensiones(cadena,&fila,&columna);
        imagen=inicializarImagen(imagen,fila,columna);
        cargarImagen(imagen,archivo);
    }
    fclose(archivo);
    return imagen;
}

```

Figura 3: Función *cargarImagenPrincipal*..

2.2.1.1 Inicializar imagen:

imagen_t inicializarImagen(imagen_t* imagen,int fila, int columna):* Esta función es utilizada para reservar memoria para la estructura *imagen_t* devolviendo un puntero a esa estructura.

```

imagen_t* inicializarImagen(imagen_t* imagen,int fila, int columna){
    imagen=(imagen_t*)malloc(sizeof(imagen_t));
    imagen->fila=fila;
    imagen->columna=columna;
    imagen->pixeles=(pixel_t**)malloc(sizeof(pixel_t)*fila);
    if (imagen->pixeles!=NULL){
        int i;
        for (i=0;i<imagen->fila;i++){
            imagen->pixeles[i]=(pixel_t*)malloc(sizeof(pixel_t)*columna);
            if (imagen->pixeles[i]==NULL){
                printf("Error: memoria insuficiente\n");
            }
        }
    }
    else{
        printf("Error: memoria insuficiente\n");
    }
    return imagen;
}

```

Figura 4: Función *inicializarImagen*.

2.2.1.2 Cargar imagen:

`void cargarImagen(imagen_t* imagen, FILE* archivo):` Luego de reservar el espacio suficiente para la imagen con la función anterior, se procede a introducir los datos leídos desde el archivo de texto. Esto se realiza recorriendo el archivo y la matriz de pixeles, de esta forma se asignan los valores r,g,b a cada pixel.

```
void cargarImagen(imagen_t* imagen, FILE* archivo){
    int c,d,e;
    int i,j;
    for (i=0;i<imagen->fila;i++){
        for(j=0;j<imagen->columna;j++){
            fscanf(archivo,"%d,%d,%d",&c,&d,&e);
            cargarPixel(c,d,e,&imagen->pixeles[i][j]);
        }
    }
}
```

Figura 5: Función cargarImagen..

2.2.2 Contar las imágenes a buscar:

`int contarImagenes():` Esta función devuelve el número de imágenes que se desean buscar en la imagen principal. Esto se lleva a cabo recorriendo línea por línea el archivo de texto “imagesBuscar.txt” que contiene todas las imágenes a buscar, y se cuenta cuantas veces aparecen dos números separados por un espacio al inicio de cada línea.

```
int contarImagenes(){
    FILE* archivo;
    archivo = fopen("imagesBuscar.txt","r");
    char cadena[LARGOLINE];
    int contadorImagenes=0;
    while (!feof(archivo)){
        fgets(cadena,LARGOLINE,archivo);
        if (isdigit(cadena[0]) && cadena[1]==' ' && isdigit(cadena[2])){
            contadorImagenes++;
        }
    }
    fclose(archivo);
    return contadorImagenes;
}
```

Figura 6: Función contarImagenes.

2.2.3 Cargar las imágenes a buscar:

`imagen_t** cargarImagenesBuscar(imagen_t** arreglo,int cantidadImagenes):` Función que es utilizada para cargar todas las imágenes que se encuentran en el archivo de texto “imagesBuscar.txt” para esto es necesario que se le entregue como parámetro un doble puntero de tipo `imagen_t` (arreglo de punteros de `imagen_t` sin inicializar) y la cantidad de imágenes (calculado con la función 2.2.2), esta función retorna un arreglo con punteros a todas las imágenes cargadas. Esta función requiere de dos funciones anteriormente mencionadas (2.2.1.1 y 2.2.1.2).

Para cargar las imágenes se recorre línea a línea el archivo y cuando encuentra dos números separados por un espacio, carga la imagen de dimensiones especificadas por los números encontrados, realizando con el mismo procedimiento que realiza la función `cargarImagenPrincipal` utilizando las dos funciones mencionadas anteriormente (2.2.1.1 y 2.2.1.2) y agrega la imagen al arreglo.

```
imagen_t** cargarImagenesBuscar(imagen_t** arreglo,int cantidadImagenes){
    arreglo=malloc(sizeof(imagen_t)*cantidadImagenes);
    FILE* archivo;
    archivo = fopen("Entrada/imagesBuscar.txt","r");
    char cadena[LARGOLINE];
    int contadorImagenes=0;
    while (!feof(archivo)){
        fgets(cadena,LARGOLINE,archivo);
        int fila,columna;
        if (buscarLinea(cadena)){
            buscarDimensiones(cadena,&fila,&columna);
            imagen_t* imagen;
            imagen=inicializarImagen(imagen,fila,columna);
            cargarImagen(imagen,archivo);
            arreglo[contadorImagenes]=imagen;
            contadorImagenes++;
        }
    }
    fclose(archivo);
    return arreglo;
}
```

Figura 7: Función `cargarImagenesBuscar`..

2.2.4 Buscar las imágenes:

`void BusquedaTotal(int cantidadImágenes,imagen_t** arregloImágenes,imagen_t* imagenPrincipal)`: La idea de este algoritmo es recorrer el arreglo de las imágenes a buscar para ir comparando cada una de las imágenes con sus rotaciones, en esta función subyacen las funciones `buscarImagen()` y la función `rotarImage()` que se explicaran a continuación:

```
void BusquedaTotal(int cantidadImágenes,imagen_t** arregloImágenes,imagen_t* imagenPrincipal){
    FILE* archivo=fopen("resultado.txt","w");
    int i;
    for (i=0;i<cantidadImágenes;i++){
        int rotar=0;
        int buscar=0;
        while (rotar<4){ //mientras no se hagan todas las rotaciones;
            buscar=buscarImagen(imagenPrincipal,arregloImágenes[i]);
            if (buscar==1){// si se encuentra la imagen sale del bucle
                fprintf(archivo,"Imagen %d: Encontrada.\n",i+1 );
                break;
            }
            if(rotar==3){
                fprintf(archivo,"Imagen %d: No encontrada.\n",i+1 );
                break;
            }
            else{
                arregloImágenes[i]=rotarImage(arregloImágenes[i]);
                rotar++;
            }
        }
    }
    fclose(archivo);
}
```

Figura 8: Función BusquedaTotal.

2.2.4.1 Buscar imagen:

`int buscarImagen(imagen_t* imagenPrincipal, imagen_t* imagen)`: Esta función recibe dos imágenes, la imagen principal y la imagen a buscar. Se recorre la imagen principal comparando el primer pixel de la imagen a buscar, si son iguales comienza a comparar el resto de pixeles, en el caso de que no sean iguales los demás pixeles, comienza de nuevo a comparar el primer pixel hasta terminar de recorrer la imagen principal.

```

int buscarImagen(imagen_t* imagenPrincipal, imagen_t* imagen){
    int i,j;
    for (i=0;i<imagenPrincipal->fila;i++){
        for (j=0;j<imagenPrincipal->columna;j++){
            if (compararPixel(imagenPrincipal->pixeles[i][j],imagen->pixeles[0][0])){
                //printf("pixel igual\n");
                int a,b,contador;
                contador=0;
                for (a=0;a<imagen->fila;a++){
                    for (b=0;b<imagen->columna;b++){
                        if (a+i<imagenPrincipal->fila && b+j<imagenPrincipal->columna){
                            if (compararPixel(imagenPrincipal->pixeles[a+i][b+j],imagen->pixeles[a][b])){
                                contador+=1;
                            }
                        }
                    }
                }
                if(contador==(imagen->fila*imagen->columna)){
                    return 1;
                }
            }
        }
    }
    return 0;
}

```

Figura 9: Función buscarImagen.

2.2.4.2 Rotar imagen:

`imagen_t* rotarImagen(imagen_t* imagen)`: esta función recibe un puntero de `imagen_t` y devuelve otro puntero de `imagen_t` con la imagen rotada. Dentro de esta función se libera el espacio utilizando por la `imagen_t` que es apuntada por el puntero de entrada.

```

imagen_t* rotarImagen(imagen_t* imagen){
    imagen_t* imagenRotada;
    imagenRotada=inicializarImagen(imagenRotada,imagen->columna,imagen->fila);
    int i,j,h;
    for (i = 0; i < imagen->fila; i++){
        h=imagen->columna-1;
        for(j = 0; j < imagen->columna; j++){
            imagenRotada->pixeles[h][i] = imagen->pixeles[i][j];
            h--;
        }
    }
    liberarImagen(imagen);
    return imagenRotada;
}

```

Figura 10: Función rotarImagen.

2.2.5 Liberar espacio utilizado

2.2.5.1 *Liberar imagen:*

`void liberarImagen(imagen_t* imagen):` Esta función realiza el proceso inverso de `inicializarImagen()`, esto se refiere a que elimina el espacio en memoria utilizado por la imagen.

```
void liberarImagen(imagen_t* imagen){
    int i;
    for (i=0;i<(imagen->fila);i++){
        free(imagen->pixeles[i]);
    }
    free(imagen->pixeles);
    free(imagen);
}
```

Figura 11: Función `liberarImagen`.

2.2.5.2 *Liberar todas las imágenes a buscar:*

`void liberarImagenesArreglo(imagen_t** arregloImagenes,int cantidadImagenes):` Esta función recorre el arreglo y va liberando todas las imágenes que se encuentran en él. Dentro de esta función subyace la función anteriormente mencionada (`liberarImagen()`).

```
void liberarImagenesArreglo(imagen_t** arregloImagenes,int cantidadImagenes){
    int i;
    for (i=0;i<cantidadImagenes;i++){
        liberarImagen(arregloImagenes[i]);
    }
}
```

Figura 12: Función `liberarImagenesArreglo..`

CAPÍTULO 3. ANÁLISIS DE LOS RESULTADOS

A continuación se muestran todas las funciones del programa con su respectivo orden:

Función	Orden
inicializarImagen	$O(n)$
liberarImagen	$O(n)$
imprimirPixel	$O(1)$
imprimirImagen	$O(n^2)$
cargarPixel	$O(1)$
cargarImagen	$O(n^2)$
compararPixel	$O(1)$
contarImagenes	$O(n^2)$
cargarImagenPrincipal	$O(n^2)$
cargarImagenesBuscar	$O(n^3)$
liberarImagenesArreglo	$O(n)$
imprimirImagenesArreglo	$O(n^3)$
rotarImagen	$O(n^2)$
buscarImagen	$O(n^4)$
busquedaTotal	$O(n^5)$

Tabla 1: Funciones y orden de algoritmo.

Como se puede apreciar la función con mayor orden es la de busquedaTotal. El orden de esta función, es mayor debido a que contiene bucles un llamado a la función buscarImagen. El orden de esta última es debido a que una matriz por cada pixel de otra matriz, teniendo 4 “for anidados desde $i=0$ hasta n ”.

CAPÍTULO 4. CONCLUSIÓN

En esta primera entrega se ha podido cumplir el objetivo, sin embargo no fue tan simple desde inicio, dado que no existía familiaridad con el lenguaje de programación, por lo que se necesitó un largo estudio antes de comenzar a programar. Lo que sirvió bastante fue abstraer el problema en subproblemas, haciendo funciones e ir revisándolas a medida que avanzaba el desarrollo de este, como también sirvió para una mejor claridad mental en el proceso.

Una de las falencias del programa es que no funciona perfectamente para imágenes muy grandes (representación de más de 3000 pixeles), debido a que lee una cierta cantidad de caracteres por línea. Esto se podría mejorar en el futuro, haciendo dinámica esa lectura dependiendo de la cantidad de caracteres en una línea.

Otra cosa que se podría mejorar es que el usuario ingrese los nombres de los archivos a trabajar, lo cual el programa debiera primero revisar si es posible trabajar con ellos, para luego proceder con la plena ejecución y entrega de resultados.

CAPÍTULO 5. REFERENCIAS

Departamento de Ingeniería Informática USACH. (2016). Enunciado Laboratorio 1. 2016: USACH.

CAPÍTULO 6. MANUAL DE USUARIO

2.3 INTRODUCCIÓN

El programa desarrollado es utilizado para verificar la existencia de imágenes, dentro de una imagen principal, las cuales pueden estar rotadas en 90, 180 y 270 grados. Estas imágenes mencionadas, deben estar en dos archivos de textos. El primero es llamado “imagenPrincipial.txt” y en él se encuentra la imagen principal. Dentro de esta se buscaran las otras imágenes contenidas en un segundo archivo nombrado “imagesBuscar.txt”. Básicamente lo que realiza el programa, es abstraer las imágenes, para luego comparar si cada pixel de las imágenes a buscar, se encuentran contenidos en la imagen principal, con la respectiva condición que estén ordenados de igual manera en ambas imágenes. Los resultados son entregados en un nuevo archivo llamado “resultado.txt”.

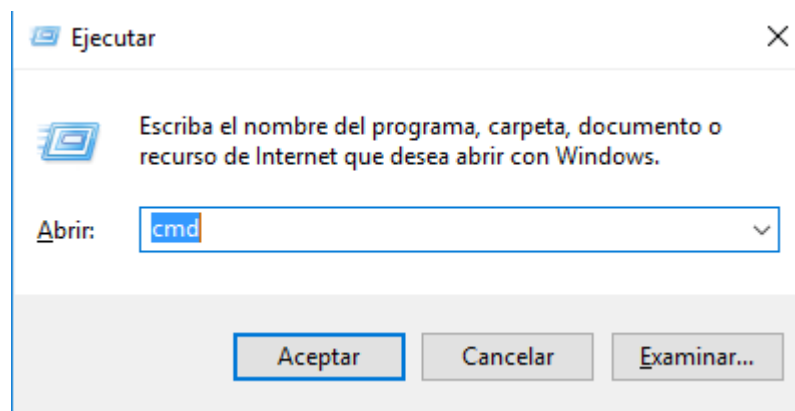
2.4 COMO COMPILAR Y EJECUTAR

Tenemos que tener en claro los siguientes comandos, que son útiles dentro de la consola de Windows y de Linux.

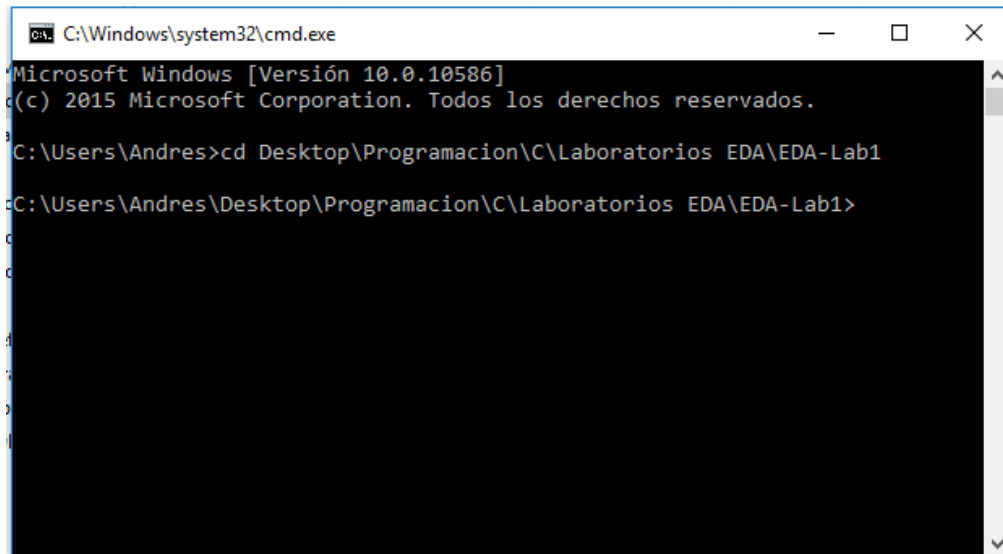
- “cd nombre_de_carpeta” para acceder a una carpeta que exista en la dirección actual
- “cd..” para volver una carpeta atras.
- “dir” para mostrar todos los archivos y carpetas existentes en la dirección actual.

2.4.1 Compilar y ejecutar en Windows:

- 1) Abrir la consola: Presionar las teclas Windows+r e ingresar cmd, luego presionar en aceptar.



- 2) Buscar la carpeta contenedora de los archivos: Se utiliza el comando `cd` seguido de la ruta de la carpeta en la que se encuentran los archivos.

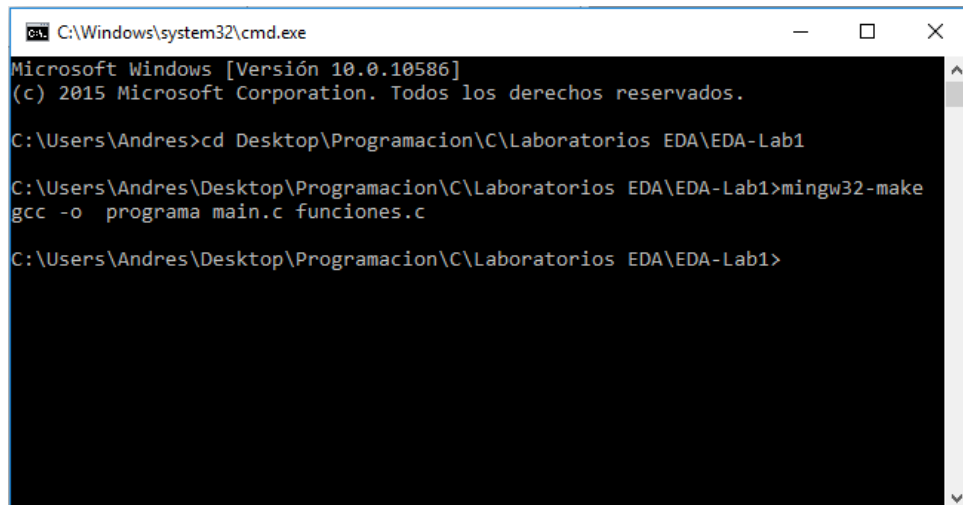


```
C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 10.0.10586]
(c) 2015 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Andres>cd Desktop\Programacion\C\Laboratorios EDA\EDA-Lab1

C:\Users\Andres\Desktop\Programacion\C\Laboratorios EDA\EDA-Lab1>
```

- 3) Compilar los archivos: Se utiliza el comando “mingw32-make”.



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 10.0.10586]
(c) 2015 Microsoft Corporation. Todos los derechos reservados.

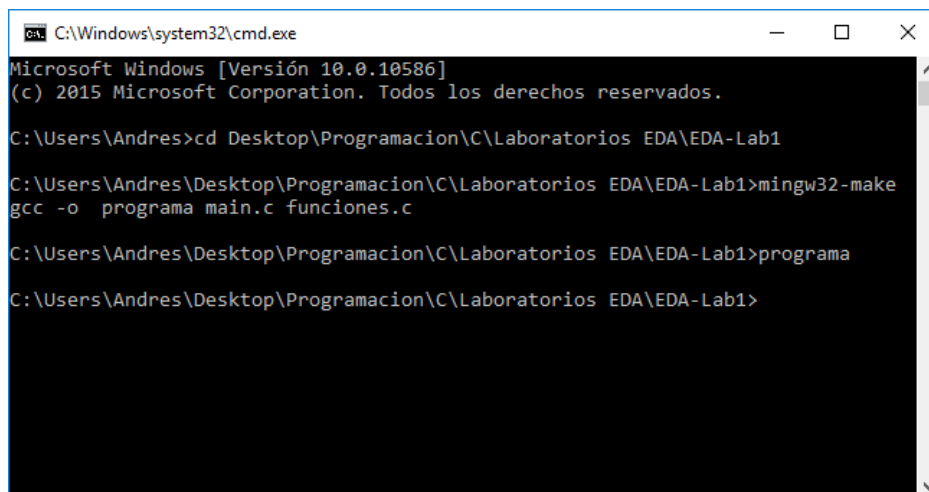
C:\Users\Andres>cd Desktop\Programacion\C\Laboratorios EDA\EDA-Lab1

C:\Users\Andres\Desktop\Programacion\C\Laboratorios EDA\EDA-Lab1>mingw32-make
gcc -o programa main.c funciones.c

C:\Users\Andres\Desktop\Programacion\C\Laboratorios EDA\EDA-Lab1>
```

Los resultados obtenidos son guardados en la carpeta “Salida”.

- 4) Ejecutar el programa: Se ingresa en la consola “programa” o doble click en el ejecutable creado al momento de compilar.



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 10.0.10586]
(c) 2015 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Andres>cd Desktop\Programacion\C\Laboratorios EDA\EDA-Lab1

C:\Users\Andres\Desktop\Programacion\C\Laboratorios EDA\EDA-Lab1>mingw32-make
gcc -o programa main.c funciones.c

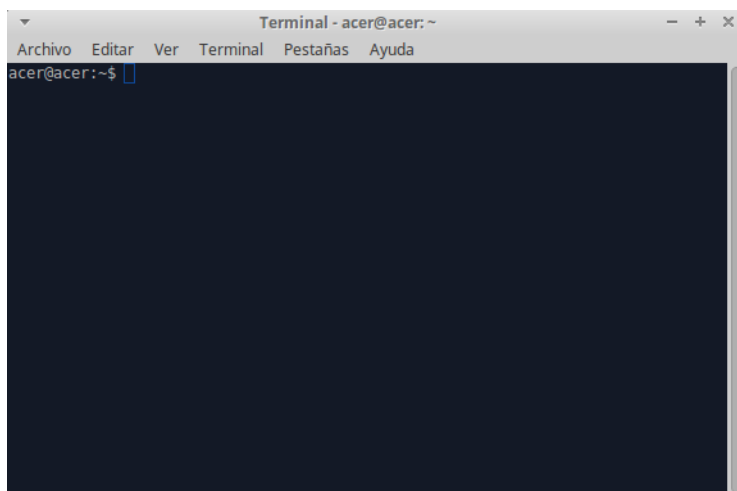
C:\Users\Andres\Desktop\Programacion\C\Laboratorios EDA\EDA-Lab1>programa

C:\Users\Andres\Desktop\Programacion\C\Laboratorios EDA\EDA-Lab1>
```

2.4.2 Compilar y ejecutar en Linux:

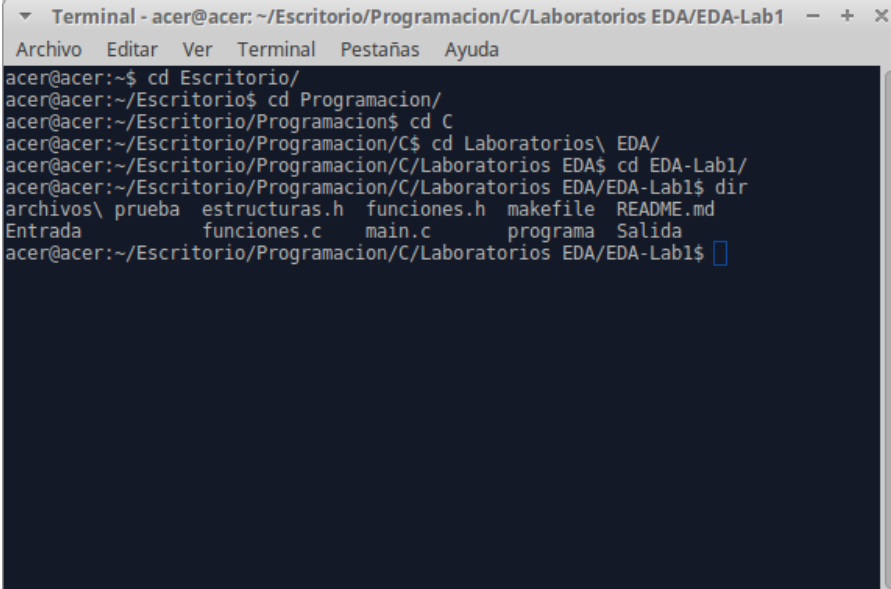
Se siguen prácticamente los mismos pasos que en Windows.

- 1) Abrir la consola: Abrir la consola o terminal desde el inicio.



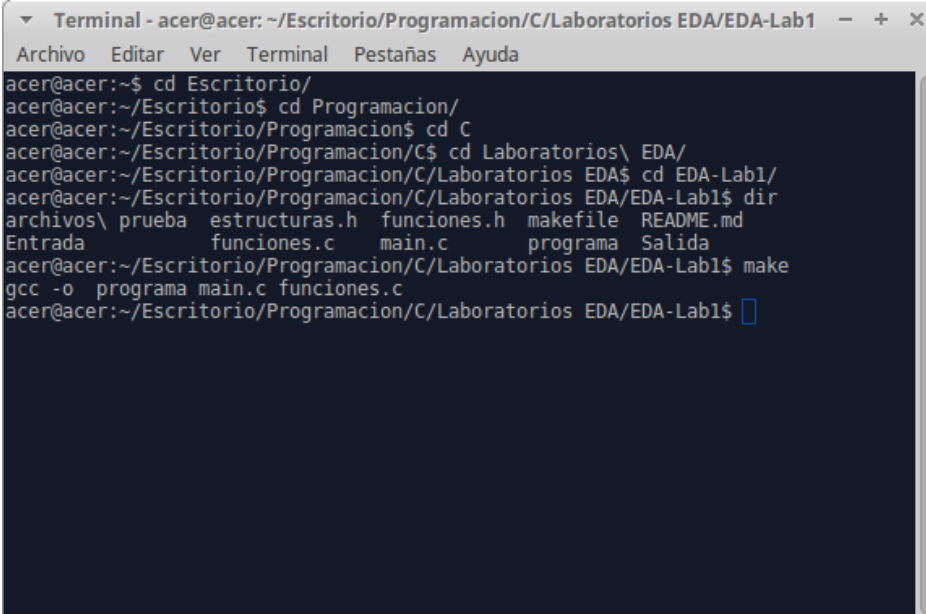
```
Terminal - acer@acer: ~
Archivo Editar Ver Terminal Pestañas Ayuda
acer@acer:~$
```

- 2) Buscar la carpeta contenedora de los archivos: Se utiliza el comando `cd` seguido de la ruta de la carpeta en la que se encuentran los archivos.



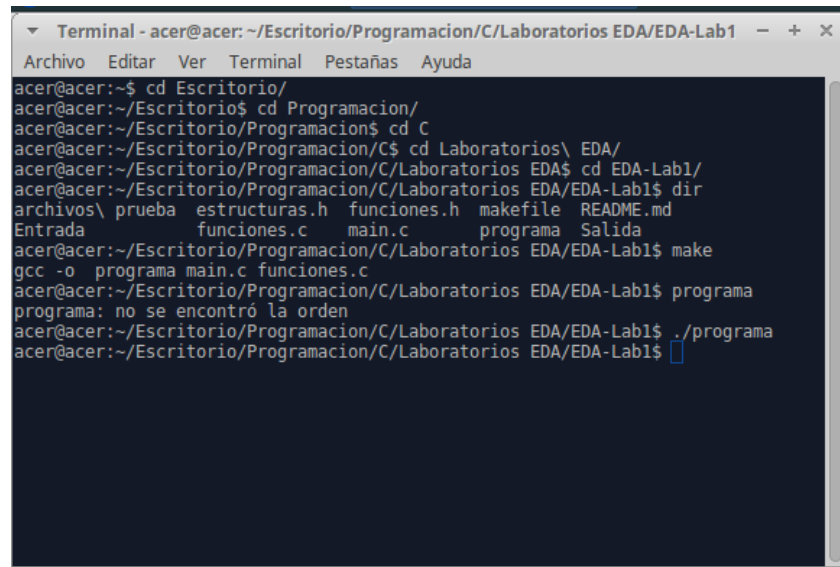
```
Terminal - acer@acer: ~/Escritorio/Programacion/C/Laboratorios EDA/EDA-Lab1 - + x
Archivo Editar Ver Terminal Pestañas Ayuda
acer@acer:~$ cd Escritorio/
acer@acer:~/Escritorio$ cd Programacion/
acer@acer:~/Escritorio/Programacion$ cd C
acer@acer:~/Escritorio/Programacion/C$ cd Laboratorios\ EDA/
acer@acer:~/Escritorio/Programacion/C/Laboratorios EDA$ cd EDA-Lab1/
acer@acer:~/Escritorio/Programacion/C/Laboratorios EDA/EDA-Lab1$ dir
archivos\ prueba  estructuras.h  funciones.h  makefile  README.md
Entrada          funciones.c  main.c      programa  Salida
acer@acer:~/Escritorio/Programacion/C/Laboratorios EDA/EDA-Lab1$
```

- 3) Compilar los archivos: Se utiliza el comando “make”.



```
Terminal - acer@acer: ~/Escritorio/Programacion/C/Laboratorios EDA/EDA-Lab1 - + x
Archivo Editar Ver Terminal Pestañas Ayuda
acer@acer:~$ cd Escritorio/
acer@acer:~/Escritorio$ cd Programacion/
acer@acer:~/Escritorio/Programacion$ cd C
acer@acer:~/Escritorio/Programacion/C$ cd Laboratorios\ EDA/
acer@acer:~/Escritorio/Programacion/C/Laboratorios EDA$ cd EDA-Lab1/
acer@acer:~/Escritorio/Programacion/C/Laboratorios EDA/EDA-Lab1$ dir
archivos\ prueba  estructuras.h  funciones.h  makefile  README.md
Entrada          funciones.c  main.c      programa  Salida
acer@acer:~/Escritorio/Programacion/C/Laboratorios EDA/EDA-Lab1$ make
gcc -o programa main.c funciones.c
acer@acer:~/Escritorio/Programacion/C/Laboratorios EDA/EDA-Lab1$
```

- 4) Ejecutar el programa: Se ingresa en la consola “./programa”. Tener cuidado en este paso ya que no sirve solo poner “programa” como en el caso de Windows. Es necesario poner “./” antes del ejecutable.



```

Terminal - acer@acer: ~/Escritorio/Programacion/C/Laboratorios EDA/EDA-Lab1
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
acer@acer:~$ cd Escritorio/
acer@acer:~/Escritorio$ cd Programacion/
acer@acer:~/Escritorio/Programacion$ cd C
acer@acer:~/Escritorio/Programacion/C$ cd Laboratorios\ EDA/
acer@acer:~/Escritorio/Programacion/C/Laboratorios EDA$ cd EDA-Lab1/
acer@acer:~/Escritorio/Programacion/C/Laboratorios EDA/EDA-Lab1$ dir
archivos\ prueba  estructuras.h  funciones.h  makefile  README.md
Entrada          funciones.c   main.c      programa  Salida
acer@acer:~/Escritorio/Programacion/C/Laboratorios EDA/EDA-Lab1$ make
gcc -o programa main.c funciones.c
acer@acer:~/Escritorio/Programacion/C/Laboratorios EDA/EDA-Lab1$ programa
programa: no se encontró la orden
acer@acer:~/Escritorio/Programacion/C/Laboratorios EDA/EDA-Lab1$ ./programa
acer@acer:~/Escritorio/Programacion/C/Laboratorios EDA/EDA-Lab1$

```

Al igual que en Windows los resultados quedan en la carpeta “Salida”.

2.5 FUNCIONALIDADES DEL PROGRAMA

La principal funcionalidad del programa, es entregar un resultado de si se encuentran las imágenes de “imagenesBuscar.txt” en “imagenPrincipal.txt” entregando como resultado un archivo de texto “resultado.txt” especificando claramente cual imagen se encontró y cuales no se encontraron. Basta con solo ejecutar el programa.

2.6 POSIBLES ERRORES

Borrar o cambiar el nombre a los archivos de textos de entrada, estos están ubicados en la carpeta “Entrada”. Esto provocara que el programa deje de funcionar.

Tener un archivo de texto que dentro contenga una imagen superior al tamaño de 3000 pixeles de ancho, puede provocar que el programa colapse, esto es debido a que el programa no está apto para leer líneas del tamaño formado por esta cantidad de pixeles.

Entregar un archivo de texto con das dimensiones incorrectas de las matrices de pixeles, puede provocar que el programa falle en su ejecución.