|  |
| --- |
| **Estructura de Datos y Análisis de Algoritmos**  **Experiencia 1: Procesamiento de imágenes** |
|  |
| **Andrés Felipe Muñoz Bravo**  **19.646.487-5** |

|  |  |
| --- | --- |
|  | Profesor: |
|  | Roberto Gonzales Ibañes |
|  |  |

|  |  |  |
| --- | --- | --- |
|  | Santiago - Chile |  |
|  | 2016 |  |

Tabla de Contenidos

[Tabla de Contenidos I](#_Toc461227028)

[Índice de Figuras II](#_Toc461227029)

[Índice de Tablas II](#_Toc461227030)

[CAPÍTULO 1. Introducción 1](#_Toc461227031)

[CAPÍTULO 2. Descripción de la solución 2](#_Toc461227032)

[2.1 Estructuras 2](#_Toc461227035)

[2.2 Funciones 3](#_Toc461227036)

[2.2.1 Cargar imagen principal: 3](#_Toc461227037)

[2.2.2 Contar las imágenes a buscar: 5](#_Toc461227038)

[2.2.3 Cargar las imágenes a buscar: 6](#_Toc461227039)

[2.2.4 Buscar las imágenes: 7](#_Toc461227040)

[2.2.5 Liberar espacio utilizado 9](#_Toc461227041)

[CAPÍTULO 3. Análisis de los resultados 10](#_Toc461227042)

[CAPÍTULO 4. Conclusión 11](#_Toc461227043)

[CAPÍTULO 5. Referencias 11](#_Toc461227044)

[CAPÍTULO 6. Manual de usuario 12](#_Toc461227045)

[2.3 Introducción 12](#_Toc461227046)

[2.4 Como compilar y ejecutar 13](#_Toc461227047)

[2.4.1 Compilar y ejecutar en Windows: 13](#_Toc461227048)

[2.4.2 Compilar y ejecutar en Linux: 15](#_Toc461227049)

[2.5 Funcionalidades del programa 17](#_Toc461227050)

[2.6 Posibles errores 17](#_Toc461227051)

Índice de Figuras

Figura 1: Estructuras…………………… pag 3

Figura 2: Función main. …………………… pag 4

Figura 3: Función cargarImagenPrincipal.. ……… pag 5

Figura 4: Función incializarImagen. ……………… pag 5

Figura 5: Función cargarImagen.. ………………… pag 6

Figura 6: Función contarImagenes. ………………… pag 6

Figura 7: Función cargarImagenesBuscar.. ……… pag 7

Figura 8: Función BusquedaTotal. …………………… pag 8

Figura 9: Función buscarImagen. …………………… pag 9

Figura 10: Función rotarImagen. …………………… pag 9

Figura 11: Función liberarImagen. …………………… pag 10

Figura 12: Función liberarImagenesArreglo.. ……… pag 10

Índice de Tablas

Tabla 1: Funciones y orden de algoritmo……..pag 11

# 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.

# 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.



## Estructuras

Para el programa se utilizan dos estructuras (pixel\_t e imagen\_t) que se muestran a continuación:

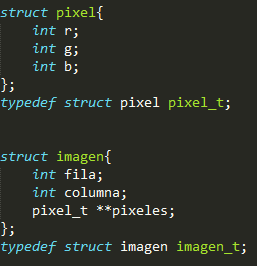


Figura 1: Estructuras

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 pixeles, en representación de una imagen. Donde “fila” es el lardo de la imagen, “columna” es el ancho de la imagen, “\*\*pixeles” representa un arreglo bidimensional (la matriz), que contiene pixeles (estructura anterior).

## 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.

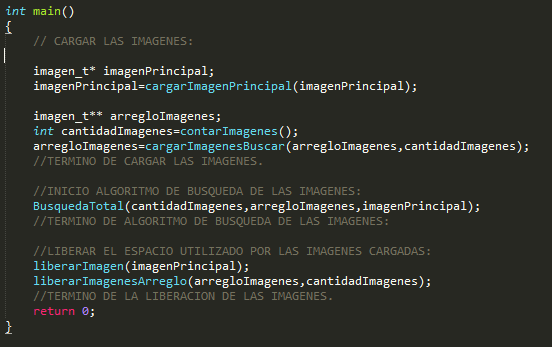


Figura 2: Función main.

### Cargar imagen principal:

*imagen\_t\* cargarImagenPrincipal(imagen\_t\* imagen):* Esta función es utilizada para introducir todos los datos de los pixeles 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.

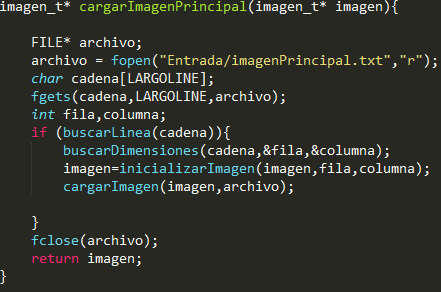


Figura 3: Función cargarImagenPrincipal..

#### **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.

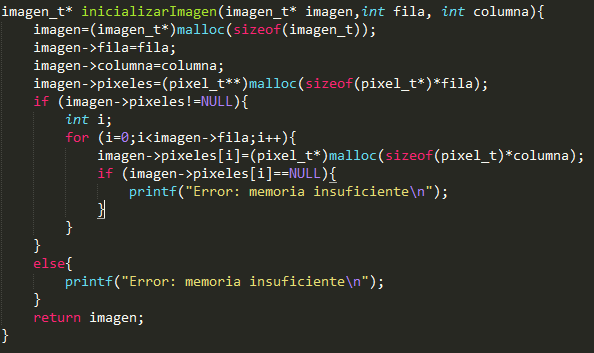


Figura 4: Función incializarImagen.

#### **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.

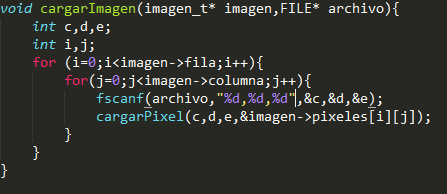


Figura 5: Función cargarImagen..

### 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.

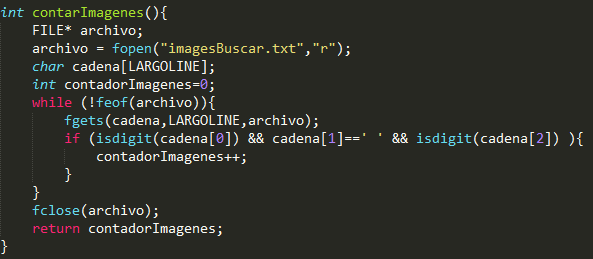


Figura 6: Función contarImagenes.

### 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.

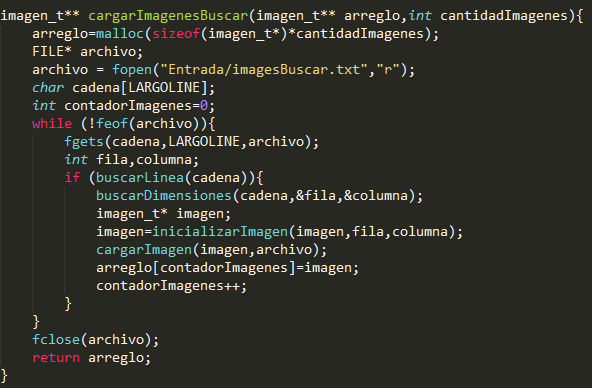


Figura 7: Función cargarImagenesBuscar..

### Buscar las imágenes:

void BusquedaTotal(int cantidadImagenes,imagen\_t\*\* arregloImagenes,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:

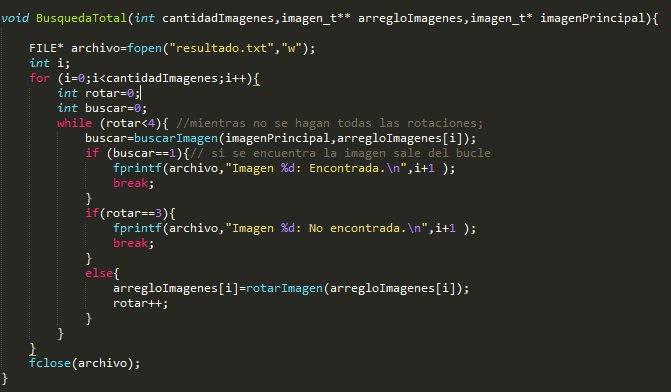


Figura 8: Función BusquedaTotal.

#### **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.

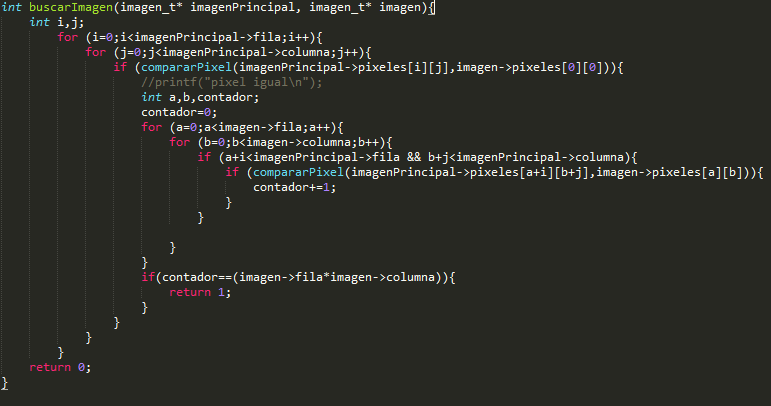


Figura 9: Función buscarImagen.

#### **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.

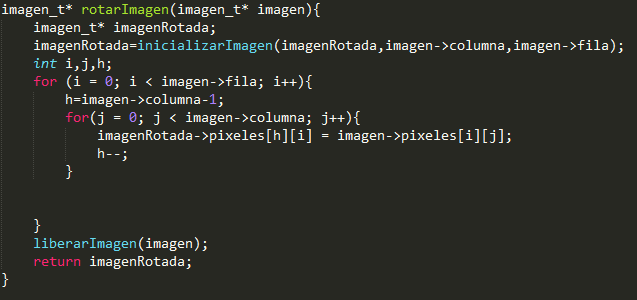


Figura 10: Función rotarImagen.

### Liberar espacio utilizado

#### **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.

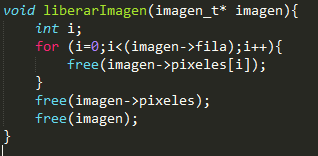


Figura 11: Función liberarImagen.

#### **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()).

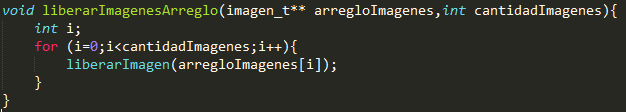


Figura 12: Función liberarImagenesArreglo..

# 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”.

# 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 avanzada 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 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.

# Referencias

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

# Manual de usuario

## 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 “imagenPrincpial.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”.

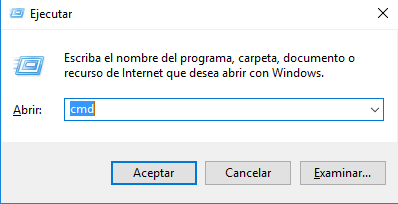
## Como compilar y ejecutar

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

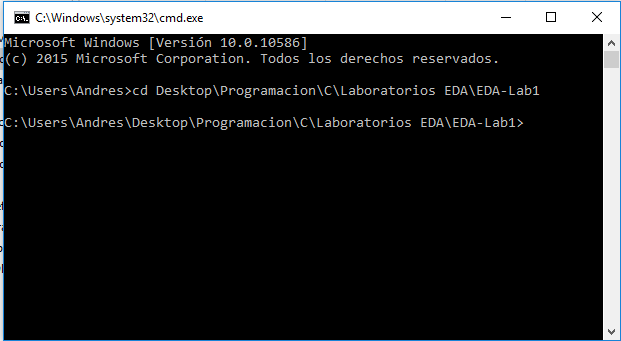
* “cd nombre\_de\_carpeta” para acceder a una carpeta que exista en la dirección actual
* “cd..” para volver una carpeta atras.
* “dir” para mostrar todos los archivos y carpetas existentes en la dirección actual.

### Compilar y ejecutar en Windows:

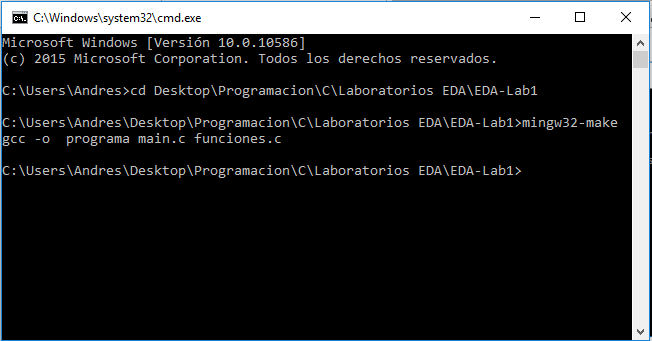
1. Abrir la consola: Presionar las teclas Windows+r e ingresar cmd, luego presionar en aceptar.



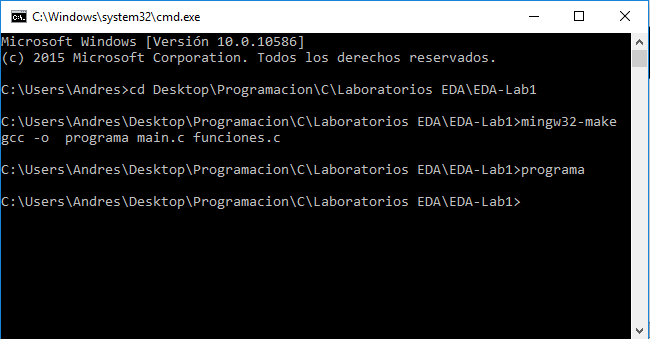
1. Buscar la carpeta contenedora de los archivos: Se utiliza el comando cd seguido de la ruta de la carpeta en la que se encuentran los archivos.



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



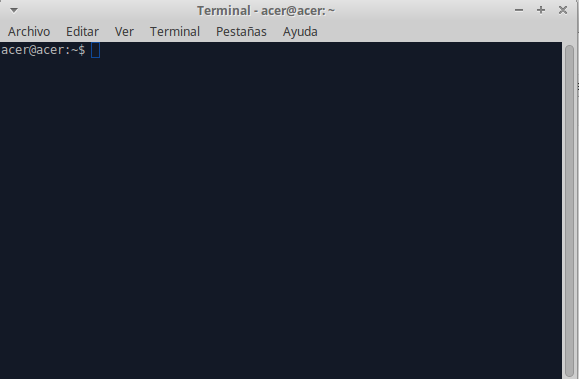
Los resultados obtenidos son guardados en la carpeta “Salida”.

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

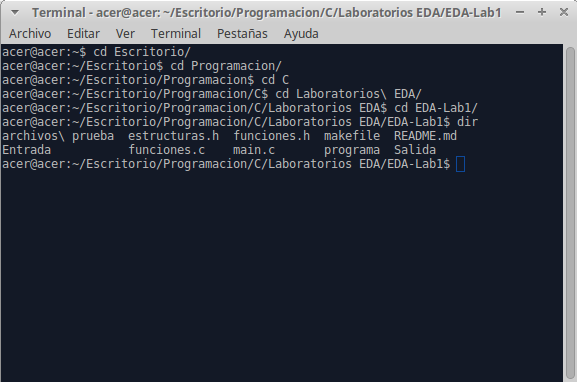
### Compilar y ejecutar en Linux:

Se siguen prácticamente los mismos pasos que en Windows.

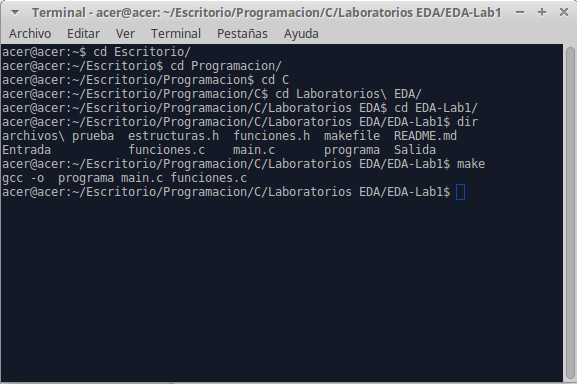
1. Abrir la consola: Abrir la consola o terminal desde el inicio.



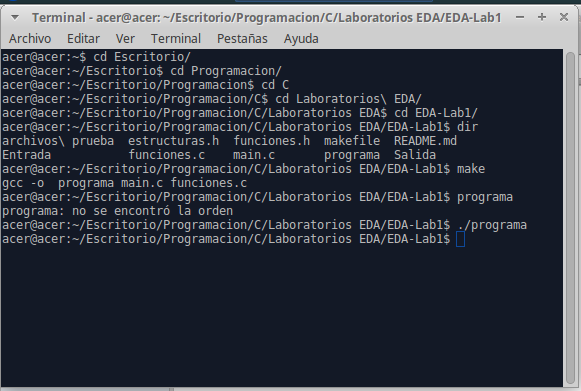
1. Buscar la carpeta contenedora de los archivos: Se utiliza el comando cd seguido de la ruta de la carpeta en la que se encuentran los archivos.



1. Compilar los archivos: Se utiliza el comando “make”.



1. 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.



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

## Funcionalidades del programa

La principal funcionalidad del programa, es entregar un resultado de si se encuentran las imágenes de “imagesBuscar.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.

## 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.