Sistemas Operativos 1/2020
Laboratorio 1

Profesores:
Cristóbal Acosta (cristobal.acosta@usach.cl)
Fernando Rannou (fernando.rannou@usach.cl)
Ayudantes:
Isaac Espinoza (isaac.espinoza@usach.cl)
Marcela Rivera (marcela.rivera.c@usach.cl)

I. Objetivos Generales

Este laboratorio tiene como objetivo refrescar los conocimientos de programación imperativa, mediante la construcción de una aplicación simple de procesamiento de imágenes. La aplicación debe ser escrita en lenguaje de programación C sobre sistema operativo Linux. Este laboratorio servirá de base a los siguientes laboratorios, los cuales irán paulatinamente incluyendo el uso de servicios del sistema operativo para realizar la misma función que la aplicación orginal. Es por eso que es de vital importancia que el diseño y modularización de esta aplicación quede bien definida a partir del lab1, para que luego sólo nos preocupemos de incluir las nuevas funcionalidades asociadas a procesos, hebras y quizá memoria compartida.

II. Objetivos Específicos

1. Conocer y usar las funcionalidades de `getopt()` como método de recepción de parámetros de entradas.
2. Construir funciones de lectura y escritura de archivos binarios usando `open()`, `read()`, y `write()`.
3. Construir funciones de procesamiento de imágenes
4. Practicar técnicas de documentación de programas
5. Conocer y practicar uso de makefile para compilación de programas.

III. Conceptos

III.A. Getopt

La función getopt pertenece a la biblioteca *<unistd.h>* y sirve para analizar argumentos ingresados por línea de comandos, asignando **options** de la forma: *"-x"*, en donde "x" puede ser cualquier letra. A continuación, se detallan los parámetros y el uso de la función:

- **int** getopt(**int** *argc*, **char * const** *argv[]*, **const char ***_optstring_);
  - *argc*: Argumento de main de tipo **int**, indica la cantidad de argumentos que se introdujeron por línea de comandos.
  - *argv[]*: Arreglo de main de tipo **char * const**, almacena los argumentos que se introdujeron por línea de comandos.
  - *optstring*: String en el que se indican los "option characters", es decir, las letras que se deben acompñar por signos "-". Si un option requiere un argumento, se debe indicar en optstring seguido de **:**, en el caso contrario, se considerará que el option no requiere argumentos y por lo tanto el ingreso de argumentos es opcional.

Getopt retorna distintos valores, que dependen del análisis de los argumentos ingresados por línea de comandos (contenidos en argv[]) y busca los option characters reconocidos en optstring.

- -1: Cuando se terminan de leer los option characters.
- ?: Cuando se lee un option no reconocido en optstring. También este es un valor de retorno cuando un option requiere un argumento, pero en línea de comandos se ingresó sin argumento.
- option character: Si en argv[] se incluyó el option "-a" y optsring contiene "a", entonces getopt retorna el char "a" y además se setea a la variable **optarg** para que apunte al argumento que acompaña al option character encontrado.

IV. Desarrollo

La aplicación consiste en un *pipeline* de procesamiento de imágenes astronómicas. Cada imagen pasará por cuatro etapas de procesamiento tal que al final del *pipeline* se clasifique la imagen como satisfaciendo o no alguna condición a definir. El programa procesará varias imágenes, una a la vez.

Las etapas del *pipeline* son:

1. Lectura de imagen RGB
2. Conversión a imagen en escala de grises
3. Filtro de realce
4. Binarización de imagen
5. Análisis de propiedad
6. Escritura de resultados

IV.A. Lectura de imágenes

Las imágenes estarán almacenadas en binario con formato jpg. Habrá n imágenes y sus nombres tendrán el mismo prefijo seguido de un número correlativo. Es decir, los nombres serán imagen_1, imagen_2, ... , imagen_n.

Para este laboratorio se leerá una nueva imagen cuando la anterior haya finalizado el *pipeline* completo. Es decir, en el *pipeline* sólo habrá una imagen a la vez.

IV.B. Conversión de RGB a escala de grises

En una imagen RGB, cada pixel de la imagen está representado por tres números, que representan el componente de color rojo (Red), el de color verde (Green) y el de color azul (Blue). Para convertir una imagen a escala de grises, se utiliza la siguiente ecuación de luminiscencia:

$$Y = R * 0.3 + G * 0.59 + B * 0.11 \tag{1}$$

donde R, G, y B son los componentes rojos, verdes y azul, respectivamente.

IV.C. Aplicar filtro laplaciano

Este tipo de filtro se basa en un operador derivativo, por lo que acentúa las zonas que tienen gran discontinuidad en la imagen. Su fórmula es:

$$\bigtriangledown^2 f = \frac{\partial^2 f}{\partial^2 x} + \frac{\partial^2 f}{\partial^2 y} \tag{2}$$

Las derivadas segundas pueden aproximarse digitalmente por:

$$\frac{\partial^2 f}{\partial^2 x} \approx f(x+1, y) + f(x-1, y) - 2f(x, y) \tag{3}$$

$$\frac{\partial^2 f}{\partial^2 y} \approx f(x, y+1) + f(x, y-1) - 2f(x, y) \tag{4}$$

La correspondiente aproximación digital del Laplaciano resulta entonces:

$$\bigtriangledown^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y) \tag{5}$$

que corresponde a la máscara espacial:

0	1	0
1	-4	1
0	1	0

Máscara de filtro Laplaciano

Figure 1. Mascara laplaciana

Lo anterior significa que el pixel [x,y] será igual a la suma total de la multiplicación de cada posición de la mascara con la respectiva posición de la imagen real.

Para los casos de borde puede mantener el valor original del pixel o bien puede rellenar con 0 el pixel faltante para aplicar la mascara normalmente.

Lo anterior se puede representar como la convolución, gráficamente el resultado de aplicar la mascara laplaciana se encuentra en la figura 3.

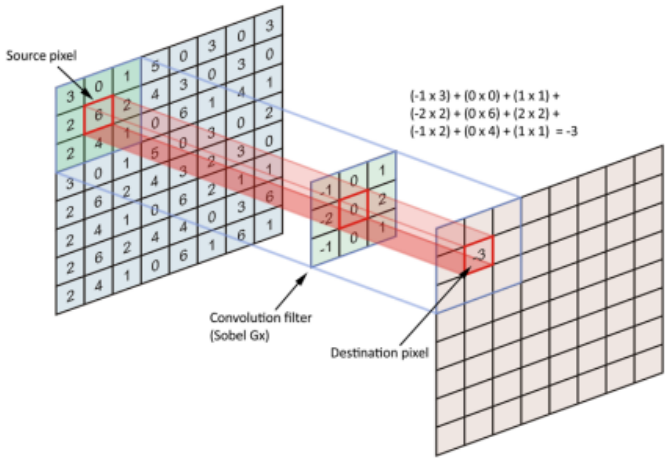

Figura 1: Operación de convolución para filtrado lineal en un punto de una imagen.

Figure 2. Implementando Mascara laplaciana

Para mayor información puede visitar el link `https://www.fceia.unr.edu.ar/dip/Filtrado_Espacial.pdf`

IV.D. Binarización de una imagen

Para binarizar la imagen basta con definir un umbral, el cual define cuáles píxeles deben ser transformados a blanco y cuáles a negro, de la siguiente manera. Para cada pixel de la imagen, hacer

```
si el pixel > UMBRAL
    pixel = 255
sino
    pixel = 0
```

Al aplicar el algoritmo anterior, obtendremos la imagen binarizada.

IV.E. Clasificación

Se debe crear una función que concluya si la imagen es *nearly black* (casi negra). Esta característica es típica de muchas imágenes astronómicas donde una pequeña fuente puntual de luz está rodeada de vacio u oscuridad. Entonces, si el porcentaje de píxeles negros es mayor o igual a un cierto umbral la imagen es clasificada como *nearly black*.

El programa debe imprimir por pantalla la clasificación final de cada imagen y escribir en disco la imagen binarizada resultante.

V. Otros conceptos

Como ayuda, la manera de leer una imagen es la siguiente:

- Abrir el archivo con el uso de open(). Recuerde que este archivo debe contener la matriz (imagen).
- Leer la imagen con read().

Con los pasos previamente descritos, se puede leer la imagen y manipularla de tal forma que se pueda crear la matriz. Recuerde que la imagen tiene una estructura para poder obtener el valor de los pixeles, por lo que se pide investigar al respecto.

Para el caso de escribir la imagen resultante:

- Se debe escribir con write().
- Finalmente se debe cerrar el archivo con close().

Por último, el programa debe recibir la cantidad de imágenes a leer, el valor del umbral de binarización, el valor del umbral de negrura y una bandera que indica si se desea mostrar o no la conclusión final hecha por la tercera función. Por lo tanto, el formato getopt es:

- -c: cantidad de imágenes
- -u: UMBRAL para binarizar la imagen.
- -n: UMBRAL para clasificación
- -m: NOMBRE del archivo que contiene la máscara a utilizar de la siguiente forma:

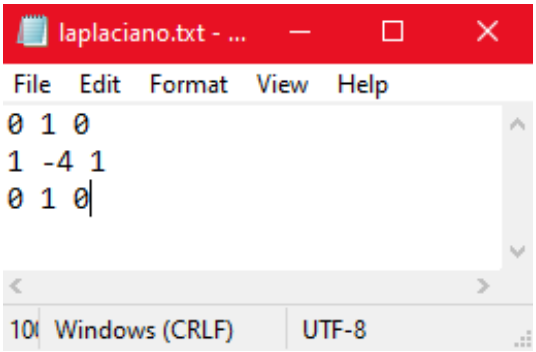

Figure 3. Ejemplo archivo máscara

Sólo se podrán recibir máscaras de 3x3, con 3 filas y 3 columnas. Las columnas separadas por espacios y las filas separadas por saltos de línea.

- -b: bandera que indica si se deben mostrar los resultados por pantalla, es decir, la conclusión obtenida al leer la imagen binarizada.

Por ejemplo, la salida por pantalla al analizar 3 imágenes sería lo siguiente:

```
$ ./pipeline -c 3 -u 50 -n 10000 -m mascara.txt -b
|     image    |     nearly black    |
|--------------|---------------------|
|   imagen_1   |         yes         |
|   imagen_2   |         no          |
|   imagen_3   |         no          |
```

Donde sólo la imagen_1 fue clasificada como *nearly black*.

VI. Entregables

Debe entregarse un archivo comprimido que contenga al menos los siguientes archivos:

1. *Makefile*: archivo make para que compile los programas.
2. uno o mas archivos con el proyecto (*.c, *.h).
3. Clara documentación, es decir, entregar códigos debidamente comentados, en donde a lo menos se incluya le descripción de cada función, sus parámetros de entrada y salida. Los comentarios deben seguir la siguiente estructura:

```
//Entradas: Explicar entradas, qué representan y su tipo de dato
//Funcionamiento: Explicación breve del funcionamiento, los comentarios específicos se debe
//Salidas: Explicar el tipo de dato de la salida y lo que representa
funcion( ... ){
//comentario especifico 1

//comentario específico 2

//...

}
```

Además, también se evalúan buenas prácticas de programación, respecto al nombramiento de variables con nombres significativos, etc.

El archivo comprimido debe llamarse: RUTESTUDIANTE1_RUTESTUDIANTE2.zip
Ejemplo: 19689333k_186593220.zip
NOTA: los laboratorios son en parejas, las cuales deben ser de la misma sección. De lo contrario no se revisarán laboratorios.

VII. Fecha de entrega

Viernes 5 de Junio. Viernes 5 de Junio.