75.04/95.12 Algoritmos y Programación II
Trabajo práctico 1: Programación C++

Universidad de Buenos Aires - FIUBA
Segundo cuatrimestre de 2014
`$Date: 2014/10/05 18:34:03 $`

1. Objetivos

Ejercitar conceptos básicos de programación C++ e implementación de TDAs. Escribir un programa en este lenguaje (y su correspondiente documentación) que resuelva el problema que presentaremos más abajo.

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá será entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes, un informe impreso de acuerdo con lo que mencionaremos en la sección 5, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

4. Introducción

En este trabajo continuaremos desarrollando nuestra herramienta para procesar imágenes, de forma tal que el programa pueda recibir funciones de transformación arbitrarias.
Con esto en mente, nuestro programa deberá poder:

- Cargar una imagen a memoria desde un archivo o desde la entrada estándar.
- Aplicar una función arbitraria, pasada por línea de comando, a la imagen.
- Guardar una imagen en memoria a un archivo o sacarlo por la salida estándar.

4.1. Interfaz

Tanto en este TP, como en el siguiente, la interacción con el programa se dará a través de la línea de comando.

Formato. Por simplicidad se usará un formato de imágenes basado en archvios de texto: *portable graymap* o PGM, con codificación ASCII[1].
Este formato define un mapa de grises: cada pixel va a tener un valor que define su intensidad entre `0` (negro) y cierto número `max` (blanco).

1. La primer línea siempre contiene `P2`, el identificador del tipo de archivo o *magic number*.

2. Luego puede haber comentarios identificados con `#` al inicio de la línea. Estos comentarios deben ser ignorados por el programa.

3. Después se presenta el tamaño de la imagen. En el ejemplo de más abajo, `24` pixels de ancho y `7` de alto.

4. Una vez definido el tamaño encontramos el máximo valor de intensidad de la imagen. En el ejemplo, `15`.

5. Por último está la imágen en sí: cada número define la intensidad de un pixel, comenzando en el margen superior izquierdo de la imagen y barriéndola por líneas hacia abajo.

Ejemplo. En el siguiente ejemplo se puede ver una imagen en formato pgm (ampliada):

Y a continuación el contenido del archivo correspondiente:

```
P2
# Shows the word "FEEP" (example from Netpbm main page on PGM)
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Transformación. Para modificar la imagen usaremos una función compleja $f : \mathbb{C} \to \mathbb{C}$. Para esto se asocia cada pixel de la imagen a un número complejo $z = a + b \cdot i$. A lo largo de este TP, vamos a suponer que la imagen cubre una región rectangular del plano complejo, cuyas coordenadas son pasadas por línea de comando. Así, los pixels de la imagen conformarán una grilla de puntos contenida dentro de esta región.

Los pixels de la imagen destino se colorean aplicando la función: para cada punto z de la imagen destino se asocia con un punto $f(z)$ en la imagen origen. Es decir, esta transformación solamente deforma la imagen original sin alterar el color del pixel.

Teniendo en cuenta las dimensiones acotadas de nuestras imágenes, se van a dar los siguientes casos:

- z pertenece a la imagen destino y $f(z)$ cae dentro de la imagen origen: este es el caso esperable.
- z pertenece a la imagen destino y $f(z)$ cae fuera de la imagen origen: asumir que z es coloreado de negro.

Este tipo de transformación permite hacer un remapeo de las imágenes. Si la función involucrada es holomorfa, se trata de una transformación conforme: la imagen transformada conserva los ángulos de la imagen original [2].

Algoritmo. En este TP, el algoritmo de evaluación de funciones a implementar es el descripto en [3]. El mismo puede ser usado para reescribir expresiones *infix* en formato RPN para luego ser evaluadas durante el proceso de transformación.

Funciones. La funciones a implementar en este TP son expresiones arbitrarias conformadas por números complejos de la forma `a + b*j`, los operadores aritméticos usuales `+`, `-`, `*`, `/`, `^`, las funciones `exp(z)`, `ln(z)`, `re(z)`, `im(z)`, `abs(z)`, `phase(z)`, y paréntesis para agrupar subexpresiones cuando sea conveniente.

Se propone a los alumnos pensar e implementar distintas funciones $f(z)$ para usar por fuera del contexto de este. Luego procesar imágenes y mandar a la lista de mails de la materia la imagen original, la procesada y la función involucrada.

4.2. Línea de comando

Las opciones `-i` y `-o` permitirán seleccionar los streams de entrada y salida de datos respectivamente. Por defecto, éstos serán `cin` y `cout`. Lo mismo ocurrirá al recibir "`-`" como argumento de cada una.

La opción `-f` permite seleccionar qué función se quiere usar para procesar la imágen. Por defecto se usará la función identidad $f(z) = z$.

Al finalizar, todos nuestros programas retornarán un valor nulo en caso de no detectar ningún problema; y, en caso contrario, devolveremos un valor no nulo (por ejemplo `1`).

Como comentario adicional, el orden de las opciones es irrelevante. Por este motivo, no debe asumirse un orden particular de las mismas a la hora de desarrollar la toma de argumentos.

4.3. Ejemplos

Primero, transformamos la imagen `grid.pgm` con la función identidad: $f(z) = z$ y guardamos la salida en `grid-id.pgm`. Ver figura 1.

```
$ ./tp0 -i grid.pgm -o grid-id.pgm -f z
```

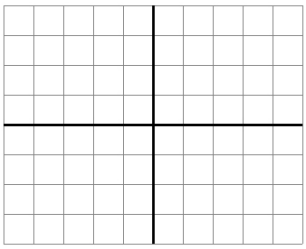

Figura 1: grid.pgm y grid-id.pgm

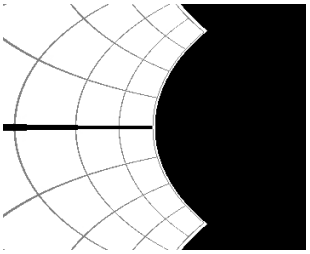

Figura 2: grid-exp.pgm

Ahora, transformamos con $f(z) = e^z$ y guardamos la salida en `evolution-exp.pgm`. Ver figuras 3 y 4).

```
$ ./tp0 -i evolution.pgm -o evolution-exp.pgm -f exp(z)
```

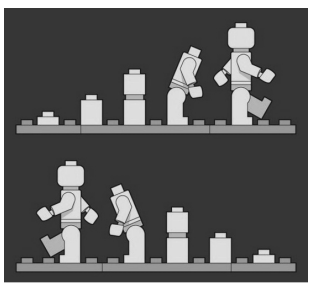

Figura 3: evolution.pgm

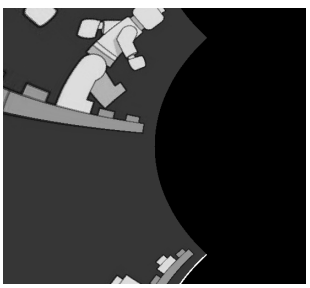

Figura 4: evolution-exp.pgm

Como siempre, estos ejemplos deben ser incluidos como punto de partida de los casos de prueba del trabajo prático.

4.4. Portabilidad

Es deseable que la implementación desarrollada provea un grado mínimo de portabilidad. Sugerimos verificar nuestros programas en alguna versión reciente de UNIX: BSD o Linux.

5. Informe

El contenido mínimo del informe deberá incluir:

- Una carátula que incluya los nombres de los integrantes y el listado de todas las entregas realizadas hasta ese momento, con sus respectivas fechas.
- Documentación relevante al diseño e implementación del programa.
- Documentación relevante al proceso de compilación: cómo obtener el ejecutable a partir de los archivos fuente.
- Las corridas de prueba, con los comentarios pertinentes.
- El código fuente, en lenguaje C++ (en dos formatos, digital e impreso).
- Este enunciado.

6. Fechas

La última fecha de entrega y presentación será el jueves 23/10.

Referencias

[1] Netpbm format (Wikipedia).
`http://en.wikipedia.org/wiki/Netpbm_format`

[2] Holomorphic function (Wikipedia).
`http://en.wikipedia.org/wiki/Holomorphic_function`

[3] Shunting yard algorithm (Wikipedia). `http://en.wikipedia.org/wiki/Shunting_yard_algorithm`.