UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA
Año 2014 – 2do Cuatrimestre

ALGORITMOS Y PROGRAMACIÓN II (75.04)

TRABAJO PRÁCTICO 0
TEMA: Programación C++
FECHA: Jueves 25 de septiembre de 2014

ENTREGADOS HASTA LA FECHA:

- TP0: 25/09/2014

INTEGRANTES:

Zaragoza, Juan Manuel - # 92.308
<juanmanuelzar@gmail.com>

Ferrari Bihurriet, Francisco - # 92.275
<fferrari@fi.uba.ar>

1 Objetivo

El principal objetivo del presente trabajo práctico consiste en implementar una herramienta para procesar imágenes, la cual recibe una imagen y una función de procesamiento para obtener una nueva imagen dependiendo de la tipo de función elegida.

El otro objetivo es ejercitar conceptos básicos de programación C++.

2 Descripción y modo de uso de la herramienta

El modo de uso de la herramienta mencionada (tp0):

```
$ ./tp0 [-i /dirección/a/archivo/entrada/*.png] [-o /dirección/a/archivo/salida/*.png] [-f funciones]
```

donde:

- /dirección/a/archivo/entrada/*.png: nombre y dirección de la imagen PNG que quiere ser procesada. En caso de no especificarse una, la herramienta adoptará la entrada estándar.
- /dirección/a/archivo/entrada/*.png: nombre y dirección de la imagen de salida procesada por la herramienta. En caso de no especificarse una, la herramienta adoptará la salida estándar.
- funciones: las funciones soportadas por la herramienta son: "z", "exp(z)", "z^2", "z^3" y "sin(z)". En caso de no especificarse una, la herramienta adoptará la la función identidad (“z”).

3 Diseño e Implementación

Se identificaron diferentes maneras de resolver el problema planteado, donde la más conveniente y sencilla de hacerlo fue creando las clases para dos de las entidades más importantes que involucran al trabajo práctico.

3.1 Problemas identificados

El primer problema que identificamos, fue cómo modelar un número complejo para aplicarle una función. Esto fue tarea sencilla debido a la implementación de la clase *complejo* que se detallará más adelante.

El segundo problema, fue cómo manipular la imagen de entrada. Para ello se creó la clase *PGMimage* la cual se encarga de leer de un archivo todo el contenido relevante a una imagen PGM para luego ser procesada en memoria, así como de escribir la imagen procesada a un nuevo archivo.

3.2 Resolución

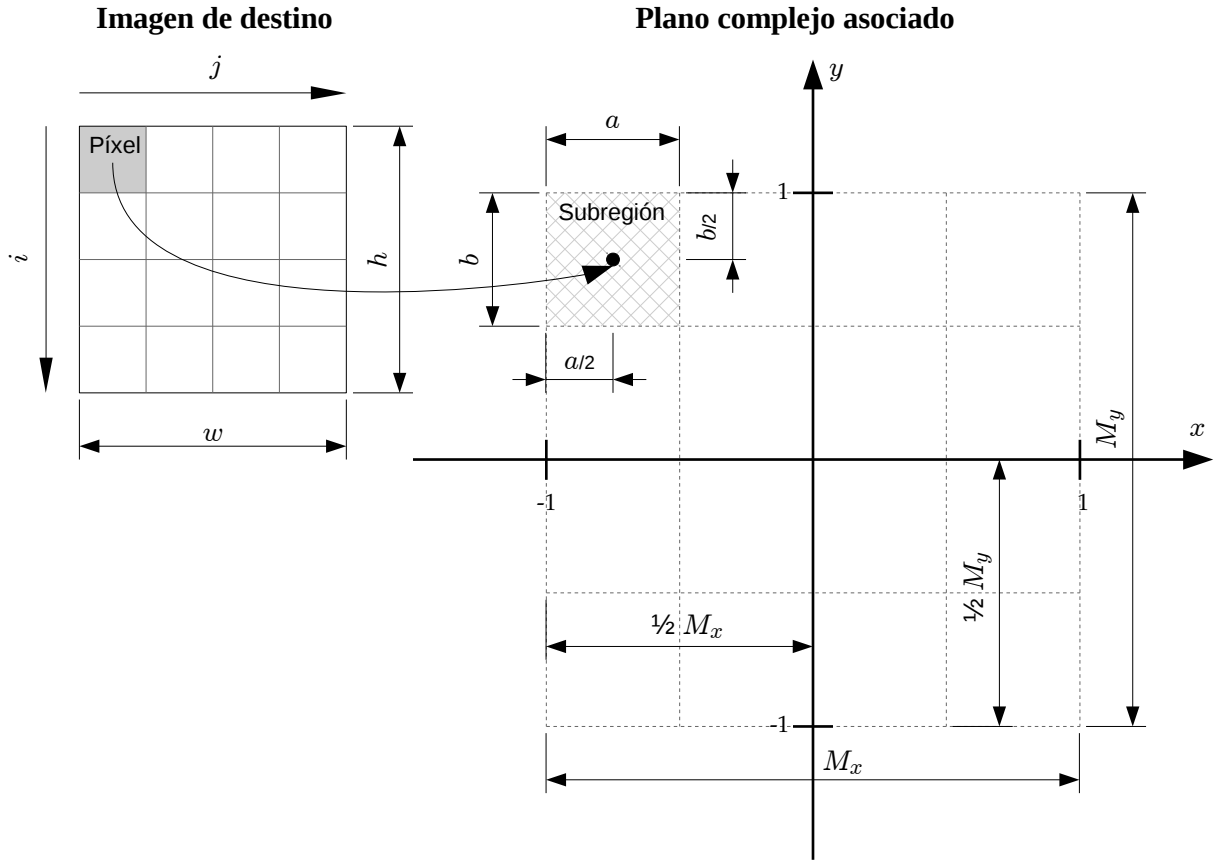
El programa principal (main) se encarga de delegar la validación de argumentos en línea de comandos, la lectura del archivo de entrada (o en su defecto, la entrada estándar), la creación de la imagen de entrada en memoria y carga con los valores del archivo, la creación

de la imagen de salida en memoria, y en cada posición de ésta, la asignación del píxel proveniente de aplicar una función compleja a la posición del píxel en la imagen de entrada, y por último, la grabación de la imagen de salida (o en su defecto, la salida estándar) con el resultado de la transformación mencionada.

3.2.1 Mapeo de la imagen al plano complejo

El último problema y más complicado de modelar, fue cómo transformar una imagen en un conjunto de puntos ubicados en el plano complejo. A continuación se detallan las transformaciones aplicadas para lograr el cometido.

Se parte de una imagen con dimensiones w x h, en una matriz indizada por i, j, para llegar a



una porción del plano complejo, centrada en el origen, de dimensiones M_x x M_y, que quedará grillada en subregiones de dimensiones a x b.

Cada subregión se corresponde con un píxel, para aplicar la función f se considerará el complejo z, correspondiente al punto central de la subregión. Para recuperar el píxel se observará en cuál subregión cae $f(z)$.

NOTA: M_x = M_y = 2, pero se utilizan las macros MAP_X y MAP_Y para aumentar la parametrización.

3.2.1.1 Mapeo de la coordenada x a partir del índice j

Nótese que $a = \frac{M_x}{w}$.

Sumaremos $a\,j + \frac{a}{2} = \frac{M_x}{w}j + \frac{M_x}{2w}$, donde el primer término es para obtener el extremo izquierdo de la subregión y el segundo para alcanzar el centro horizontal de la misma.

A esto hay que restarle $\frac{M_x}{2}$ para centrar todo en el origen del plano complejo, resultando en $\frac{M_x}{w}j + \frac{M_x}{2w} - \frac{M_x}{2}$, que sacando factores comunes resulta en:

$$\boxed{x = M_x\left(\frac{j+0{,}5}{w} - 0{,}5\right)}$$

3.2.1.2 Mapeo de la coordenada y a partir del índice i

Nótese que $b = \frac{M_y}{h}$.

Sumaremos $b\,i + \frac{b}{2} = \frac{M_y}{h}i + \frac{M_y}{2h}$, donde el primer término es para obtener el extremo inferior de la subregión y el segundo para alcanzar el centro vertical de la misma.

A esto hay que restarle $\frac{M_y}{2}$ para centrar todo en el origen del plano complejo, resultando en $\frac{M_y}{h}i + \frac{M_y}{2h} - \frac{M_y}{2}$, pero se advierte que la dirección de i es la inversa a la de y, por lo que habrá que hacer un cambio de signo, resultando en:

$$\boxed{y = M_y\left(0{,}5 - \frac{i+0{,}5}{h}\right)}$$

3.2.1.3 Proceso inverso: complejo en subregión hacia índices

Si se despejan las inversas de las funciones detalladas en las secciones anteriores, se obtiene:

$$j = w\left(0{,}5 + \frac{x}{M_x}\right) - 0{,}5 \quad ; \quad i = h\left(0{,}5 - \frac{y}{M_y}\right) - 0{,}5$$

Si se redondean estos números, se estarán obteniendo los índices correspondientes para cada valor de $f(z)$ dentro de las subregiones. Una forma de redondear es sumar 0,5 y truncar al valor entero (función “piso” o “*floor*”). Se aprovechará esto para deshacerse del 0,5 que resta en ambas expresiones, por lo que finalmente se llega a:

$$\boxed{j = \lfloor w\left(0{,}5 + \frac{x}{M_x}\right)\rfloor} \; ; \; \boxed{i = \lfloor h\left(0{,}5 - \frac{y}{M_y}\right)\rfloor}$$

NOTA: el comportamiento de C++ al convertirse de *float* a *int* es el truncamiento al valor entero.

3.2.2 Clase Complejo

La clase complejo se encarga de abstraer y modelar la implementación y manipulación de números complejos. Posee dos atributos: parte real y parte imaginaria.

3.2.2.1 Constructores

La clase posee tres constructores: uno que crea un objeto con valores por defecto (0.0 tanto para la parte real como para la imaginaria), un constructor por copia que recibe un complejo y un constructor que recibe dos *doubles* y los setea en la parte real e imaginaria del objeto creado.

3.2.2.2 Sobrecarga de operadores

La clase sobrecarga los siguientes operadores:

- operador <<, imprime un complejo del siguiente modo “(x,y)” a través del *ostream*. Modo de uso: **std::cout << c;** donde **complejo c = complejo(1.2,3.0);**
- operador >>, lee un complejo desde el *istream* con el formato “(x,y)”. Modo de uso: **std::cin >> c;**
- operador =, setea la parte real e imaginaria con los valores del complejo pasados como argumento. Modo de uso: **b = c;** donde **complejo c = complejo(1.2,3.0);** y **complejo b;**
- operator +, suma dos complejos o un complejo con un número real. Modo de uso: **b = b + c;** donde **complejo c = complejo(1.2,3.0);** y **complejo b(1.0, 1.0);**

3.2.2.3 Getters y Setters

Estos métodos se encargan de fijar/obtener los atributos del objeto complejo. Los atributos en cuestión son la parte real y la parte imaginaria.

3.2.2.4 Métodos de funciones

Las funciones son las encargadas de aplicarle una operación matemática a un número complejo y obtener el resultado complejo. Las funciones implementadas son: f(z) = z, f(z) = e^z, f(z) = z^2, f(z) = z^3, f(z) = sin(z).

3.2.3 Clase PGMimage

Esta clase representa una imagen extraída de un archivo PGM en memoria. Posee los siguiente atributos: MagicNumber, width (ancho de la imagen), height (alto de la imagen), ColorDepth (profundidad de color) y canvas (lienzo en memoria).

3.2.3.1 Constructores

La clase posee dos constructores implementados: un constructor con argumentos por defecto (ancho y alto igual a 1, y profundidad de color igual al máximo permitido por la aplicación) y un constructor por copia que recibe una PGMimage. El primero recibe ancho, alto y profundidad del color, limitando este último, crea un nuevo lienzo en memoria y setea cada uno de los valores del 0 a negro. El segundo, recibe una PGMimage, copia los valores de ancho, alto, profundidad y los píxeles del lienzo.

3.2.3.2 Destructores

La clase implementa un destructor que libera la memoria utilizada para almacenar el lienzo.

3.2.3.3 Sobrecarga de operadores

La clase sobrecarga los siguientes operadores:

- operador [], se encarga de obtener cada píxel de la imagen en la posición i,j. Modo de uso: **out_image[i][j] = in_image[row][col];** donde **PGMimage out_image (w, h, d);**
- operador >>, se encarga de cargar la imagen en memoria desde flujo/archivo/stdin. Modo de uso: ***in_stream >> in_image;** donde **PGMimage in_image;** y **istream *in_stream = &cin;**
- operador >>, se encarga de imprimir la imagen que se encuentra en memoria en el flujo/archivo/stdin con el formato de archivo PGM. Modo de uso: ***out_stream << out_image;** donde **PGMimage out_image;** y **ostream *out_stream = &cout;**

3.2.3.4 Getters y Setters

Estos métodos se encargan de fijar/obtener los atributos del objeto PGMimage.

Los métodos getWidth() , getHeight() y getColorDepth() se encargan de obtener el ancho, alto y profundidad del archivo respectivamente.

El método setColorDepth(pixel_t) se encarga de fijar la profundidad de colores de la imagen.

El método resize(size_t, size_t) se encarga de modificar el tamaño del lienzo con un nuevo ancho y alto. No se realiza un escalado de la imagen, solo se modifica el lienzo, resultando en posibles recortes o agregado de píxeles con eventual basura.

3.2.3.5 Utilidades internas

Existen tres utilidades internas que son importantes para el uso dentro de los métodos de la clase.

- static void ignore_comments(istream &); recibe el flujo/archivo e ignora los comentarios (empezados con '#').
- static pixel_t** new_canvas(size_t, size_t); reserva memoria para crear el lienzo en memoria.
- void canvasDestroy(); destruye el lienzo creado en memoria.

3.2.4 Programa principal

El programa principal realiza los siguientes pasos:

1. Por línea de comando se validan los argumentos y se setean los flujos/archivos de donde se leerán y/o escribirán las imágenes PGM.
2. Se abren los archivos en caso de no haberse elegido las entradas/salidas estándar.
3. Se crea una nueva imagen en memoria, se lee desde el flujo/archivo la imagen y se sube a memoria.
4. Se crea una nueva imagen en memoria pero de salida con los tamaños y profundidad de la imagen de entrada.
5. Se crean dos complejos con constructores por defecto.
6. Se comienza a recorrer la imagen de salida, a los índices se le aplica la función elegida por línea de comandos y se obtiene la posición en el lienzo de entrada.
7. El píxel del lienzo de entrada se guarda en la posición de la nueva imagen.
8. Por último, el lienzo de salida que se encuentra en memoria se escribe en el flujo/archivo de salida.

4 Proceso de compilación

Para compilar los archivos en entornos *unix-like*, fue diseñado el *Makefile*, con las siguientes opciones:

➔ Para generar todos los archivos objeto, más el ejecutable en el directorio ./bin/

```
$ make
```

Este comando crea el directorio ./bin/ compilando el código fuente para obtener los siguientes archivos en código objeto:

- PGMimage.o: Clase para manejo de imágenes PGM.
- complejo.o: Clase para manejo de número complejos.
- utils.o: Utilidades del programa, como las funciones internas y los parsers de CLA.
- cmdline.o: Manejo de CLA, provisto por los docentes, levemente modificado.
- main.o: Programa principal, propiamente dicho.

Luego estos archivos son "*linkeados*" en el ejecutable ./bin/tp0

➔ Para "limpiar" (eliminar todos los binarios, y el directorio ./bin/)

```
$ make clean
```

➔ Para eliminar los archivos objeto

```
$ make objclean
```

➔ Para compilar eliminando los archivos temporarios

```
$ make deltemps
```

➔ Para instalar (*)

```
# make install
```

➔ Todo: compilar, instalar, limpiar (*)

```
# make all
```

➔ Desinstalar (*)

```
# make remove
```

➔ Desinstalar y limpiar (*)

```
# make purge
```

(*): Se requieren permisos de superusuario

NOTA: también fue probado en *Windows* con *MinGW*, donde las utilidades extra no funcionan, pero si el comando make a secas.

5 Ejecución y resultados

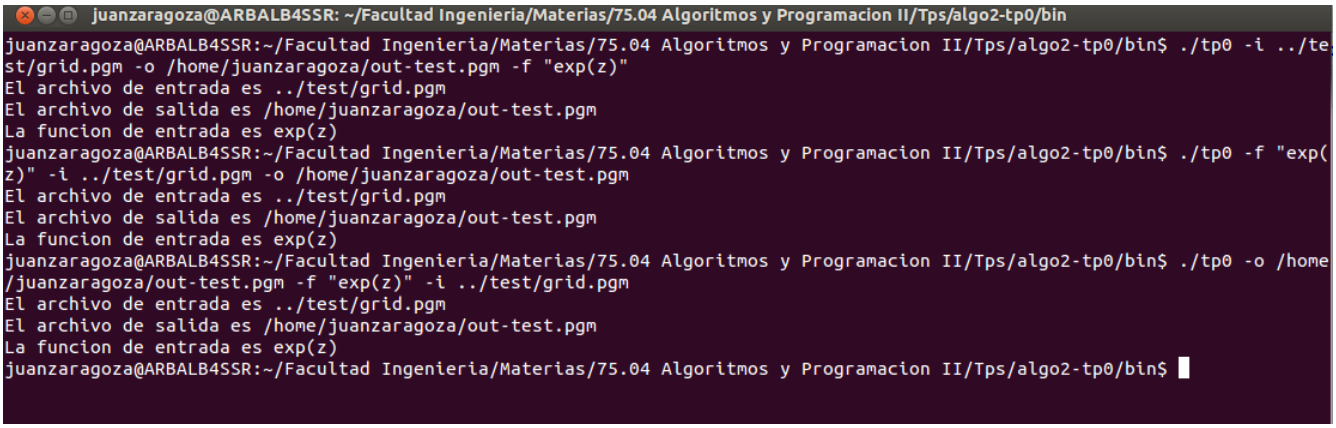
En está sección se describirán los diferentes modo de uso y resultados de las sucesivas pruebas realizadas.

5.1 Modos de ejecución

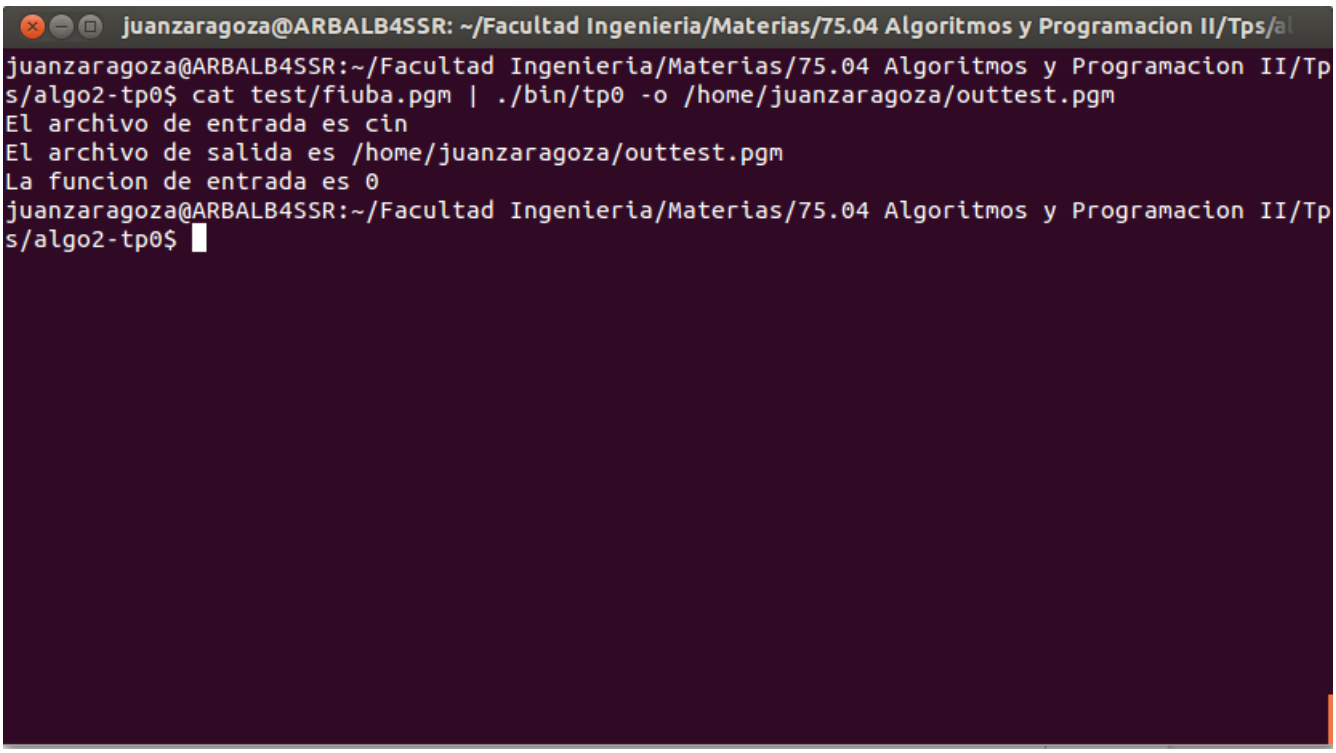
Existen diferentes modo de ejecución de la herramienta. A fines de mostrar los resultados, se han dejado algunos mensaje informativos acerca de la consola de comandos.

1. Si ejecutamos la herramienta sin ningún tipo de argumentos, el sistema tomará las entrada/salida/función por defecto. Al no especificarse un archivo ni ingresarse un archivo en la entrada estándar, el sistema devuelve un error.

2. La herramienta identifica los parámetros pasados sin importar el orden de los mismos.



3. Para usar la entrada estándar, la herramienta puede utilizarse de la siguiente manera. Tener en cuenta que si no es especifica un archivo de salida el resultado se imprimirá en consola, lo cual resultará insignificante si queremos ver algún resultado.



5.2 Resultados

Se mostrarán los diferentes resultados obtenidos al correr diferentes pruebas. Notar que no es necesario hacer una captura de la consola luego de ejecutarse el comando, por lo tanto, solo mostraremos la imagen origen y destino junto al comando ejecutado.

Primero mostraremos la imagen con la función identidad para entender cual es la imagen original y luego mostraremos la imagen transformada.

Figura 1: $ ./tp0 -i grid.pgm -o grid-id.pgm -f z

Figura 2: $ ./tp0 -i grid.pgm -o grid-exp.pgm -f “exp(z)”

Figura 3: $ ./tp0 -i grid.pgm -o grid-z2.pgm -f “z^2”

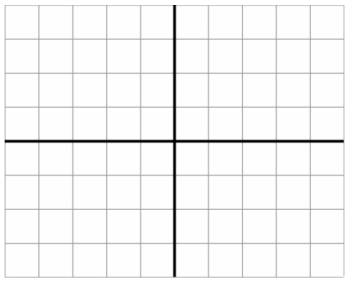


Figura 1: grid-id.pgm

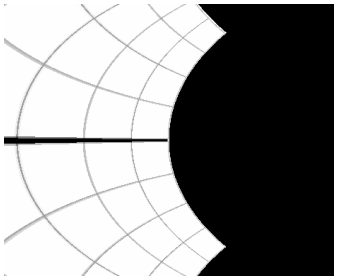


Figura 2: grid-exp.pgm

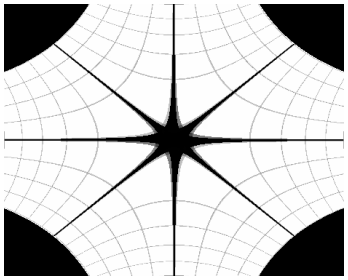


Figura 3: grid-z2.pgm

Figura 4: $ ./tp0 -i toys.pgm -o toys-id.pgm

Figura 5: $ ./tp0 -i toys.pgm -o toys-exp.pgm -f “exp(z)”

Figura 6: $ ./tp0 -i toys.pgm -o toys-sin.pgm -f “sin(z)”

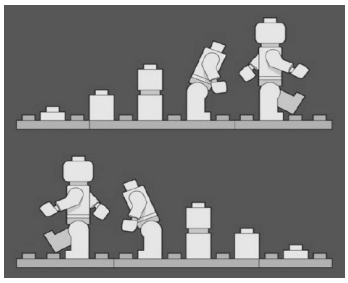


Figura 4: toys-id.pgm

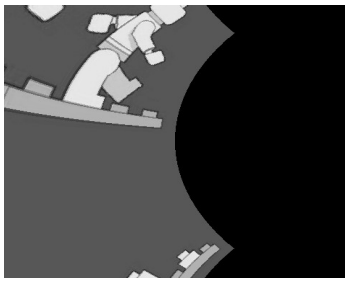


Figura 5: toys-exp.pgm

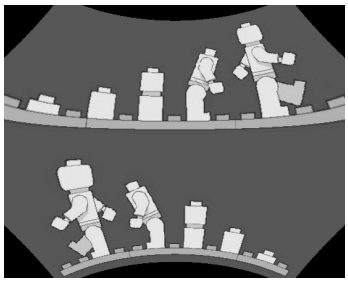


Figura 6: toys-sin.pgm

Figura 7: $ ./tp0 -i fiuba.pgm -o fiuba-id.pgm

Figura 8: $ ./tp0 -i fiuba.pgm -o fiuba-exp.pgm -f “exp(z)”

Figura 9: $ ./tp0 -i fiuba.pgm -o fiuba-z3.pgm -f “z^3”

Figura 7: fiuba-id.pgm

Figura 8: fiuba-exp.pgm

Figura 9: fiuba-z3.pgm

6 Conclusión

El trabajo práctico presentado, nos originó diferentes dificultades. Al principio se nos hizo difícil entender cuál era el propósito del mismo por un tema de comprensión de las especificaciones descriptas (¿qué es lo que debería hacer el programa?). Luego, fue complicado darse cuenta como modelar lo que se estaba pidiendo (modelar un conjunto de índices y transformarlos a números complejos; ídem. inversa). Por último, llevó mucho trabajo poder programar lo que habíamos modelado.

Es un trabajo interesante para poder aprender algunas técnicas y conceptos de C++, tales como el uso de clases e instanciación de objetos.

A esto se suma lo interesante de la transformación de imágenes mediante funciones holomorfas, haciendo más amena la tarea de resolver las dificultades.

7 Bibliografía

- Netpbm format (Wikipedia).
  http://en.wikipedia.org/wiki/Netpbm_format

- Holomorphic function (Wikipedia).
  http://en.wikipedia.org/wiki/Holomorphic_function

- Conformal pictures (Wikipedia).
  http://en.wikipedia.org/wiki/Conformal_pictures

- Standard C++ Library reference (cplusplus.com).
  http://www.cplusplus.com/reference/