Manual de Referencia de PNGwriter

Versión 0.5.3 (24 / I / 2004)

http://pngwriter.sourceforge.net/

Introducción

Este es el Manual de Referencia Rápida de PNGwriter. Es un resumen de las funciones de PNGwriter. Para una descripción más detallada, ver el archivo header `pngwriter.h`, aunque este manual es bastante completo. Asegúrate de revisar los ejemplos en el website, y los que encontrarás en `/usr/local/share/doc/pngwriter/`.

Nota: Éste documento asume que PNGwriter fue instalado en `/usr/local` (en `/usr/local/lib`, `/usr/local/share`, etc.), lo que sucede por defecto. Si elegiste otro lugar para instalarlo, entonces ten esto en cuenta al momento de leer este Manual. Esto es importante para saber qué directorios de librerías y headers hay que especificar al momento de compilar, dónde encontrar la documentación y ejemplos, dónde encontrar los fonts incluidos, etc. Por ejemplo, si instalaste PNGwriter desde el paquete de Debian, el directorio bajo el cual se instala es `/usr` (en `/usr/lib`, `/usr/share`, etc.). O, si al instalarlo desde el código fuente, diste el comando `make install PREFIX=$HOME` por ejemplo, donde `$HOME` es la ubicación de tu directorio personal, encontrarás los archivos de PNGwriter instalados en `$HOME/lib`, `$HOME/share`, etc.

Resumen

PNGwriter es una librería gráfica que usa el formato PNG. La interface ha sido diseñada para ser lo más simple e intuitiva posible. Permite plotear y leer en los espacios de colores RGB (rojo, verde, azul), HSV (hue, saturation, value/brightness) y CMYK (cyan, magenta, amarillo, negro), firguras básicas, escalamiento de imágenes, interpolación bilineal, soporte completo para texto TrueType rotado y con antialiasing, curvas bezier, abrir imágenes PNG ya existentes y más. Documentación en inglés y español. Funciona en Linux, Unix, Mac OS X y Windows. Requiere libpng y opcionalmente FreeType2 para el soporte de texto.

Contenido

Este documento está dividido en las siguientes secciones principales:

Ayuda y Soporte

Si tienes un problema, una pregunta o sugerencia, puedes enviar un mensaje o unirte a la lista de correo de PNGwriter. Preferiría que hicieras esto último, ya que mi respuesta te llegará de inmediato, y podras estar al día con las nuevas versiones de PNGwriter. También me puedes mandar un email directamente.

La documentación de PNGwriter consiste en este Manual PDF, el archivo LEAME, el archivo de encabezado `pngwriter.h`, los dos ejemplos completos incluídos con el código fuente y, por supuesto, el sitio web, el cual tiene una sección de Preguntas Frecuentes y algunos ejemplos bastante interesantes, entre otras cosas.

Notas Generales

Uso Básico

El uso más simple de PNGwriter sería lo siguiente:

```
#include <pngwriter.h>

int main()
{
     pngwriter image(200, 200, 1.0, "out.png");
     image.plot(30, 40, 1.0, 0.0, 0.0);
     image.close();

     return 0;
}
```

Esto produce un punto rojo (1 pixel) en la esquina inferior izquierda de la imagen, sobre un fondo blanco. El archivo que se creará al correr el programa se llamará “out.png”.

Para comenzar a usar PNGwriter, lee acerca del constructor, plot(), dread() y close(). Explora las otras funciones cuando te sientas cómodo con éstas.

Es importante recordar que todas las funciones que aceptan un argumento del tipo “`const char *`“ también aceptarán “`char *`“. Esto es así para poder tener un nombre de archivo cambiante (para crear varias imágenes PNG en serie con nombres distintos, por ejemplo), y para permitir el uso de objetos de tipo string, los que pueden ser fácilmente convertidos en `const char *` (si `elString` es un objeto de tipo string, entonces puede ser usado como un `const char *` llamando a `elString.c_str()`).

Tambien es importante recordar que cuando una función tiene un coeficiente de color como argumento, ese argumento puede ser un `int` de `0` a `65535` o un `double` de `0.0` a `1.0`. Debes asegurarte que estás llamando a la función adecuada. Recuerda que 1 es un int, mientras que 1.0 es un double, y esto determinará la versión de la función que usarás. No cometas el error de llamar por ejemplo a `plot(x, y, 0.0, 0.0, 65535)`, porque no hay un plot(int, int, double, double, int). Además, nota que `plot()` y `read()` (y las funciones que las usan internamente) están protegidas contra el uso de argumentos negativos o fuera de rango. `read()` devolverá `0` cuando se llame fuera del rango de la imagen. Esto incluso puede ser útil como ‘zero-padding’ si es que lo necesitaras.

Compilación

Una compilación típica, asumiendo que PNGwriter fue instalado en su ubicación por defecto (bajo `/usr/local`), se vería así.

```
g++ my_program.cc -o my_program `freetype-config --cflags`
     -I/usr/local/include  -L/usr/local/lib -lpng -lpngwriter -lz -lfreetype
```

Si no fue instalado ahí, reemplaza `/usr/local` en las líneas anteriores por la ubicación que elegiste al momento de instalar.
Si no compilaste PNGwriter con soporte de FreeType, entonces quita las opciones que tengan que ver con FreeType y agrega `-DNO_FREETYPE`.

El website tiene información adicional acerca de cómo compilar en Windows.

Constructor

El constructor requiere el ancho y la altura de la imagen, el color de fondo para la imagen, y el nombre del archivo (un puntero o simplemente "`miarchivo.png`"). El constructor por defecto crea una instancia de PNGwriter de `250x250` pixeles, fondo blanco y nombre "out.png". **Sugerencia**: el nombre del archivo puede ser especificado tan fácilmente como escribir `mypng(300, 300, 0.0, "myfile.png");`. **Sugerencia**: Si vas a crear una instancia de PNGwriter para abrir con ella una imagen PNG ya existente, entonces la altura y el ancho pueden ser `1` pixel, y el tamaño quedará determinado automáticamente luego de abrir el PNG con readfromfile().

```
pngwriter();
pngwriter(    const pngwriter &rhs);
pngwriter(    int ancho, int altura, int backgroundcolour, char * filename);
pngwriter(    int ancho, int altura, double backgroundcolour, char *
              filename);
pngwriter(    int ancho, int altura, int backgroundcolour, const char *
              filename);
pngwriter(    int ancho, int altura, double backgroundcolour,
              const char * filename);
```

Operador de Asignación

PNGwriter sobrecarga el operador de asignación =.

```
pngwriter & operator = (const pngwriter & rhs);
```

Versión

Da la versión de PNGwriter.

```
double version(void);
```

. . . . . Plotear . . . . . .

Plotear

Los pixeles se enumeran partiendo desde `(1, 1)` y van hasta (`ancho`, `altura`). Al igual como sucede con muchas funciones en PNGwriter, esta función ha sido sobrecargada para aceptar argumentos tipo int o double para

los coeficientes de color. Si son de tipo `int`, van desde `0` a `65535`. Si son de tipo `double`, van desde `0.0` hasta `1.0`. **Sugerencia**: Para plotear usando rojo, escribe `plot( x, y, 1.0, 0.0, 0.0).` Para hacer rosado, agrega una constante a los tres coeficientes: `plot( x, y, 1.0, 0.4, 0.4).` **Sugerencia**: Si no obtienes nada en tu imagen PNG, acérdate de llamar a la función close() de la instancia antes que tu programa termine, y que la posición x e y de los pixeles que ploteas esté dentro del rango de tu imagen. Si no lo están, entonces PNGwriter no se quejará-- es responsabilidad tuya asegurar que esto no suceda. **Sugerencia**: Si tratas de plotear con un coeficiente de color fuera de rango, se asumirá un coeficiente de color máximo o mínimo, de acuerdo al coeficiente usado. Por ejemplo, el intentar plotear `plot(x, y, 0.5, -0.2, 3.7)` lo hará usando `0.0` para el coeficiente verde y `1.0` para el coeficiente azul.

```
void plot(int x, int y, int rojo, int verde, int azul);
void plot(int x, int y, double rojo, double verde, double azul);
```

Plotear Combinado

Plotea el color dado por `red`, `green`, `blue`, pero combinado con el valor del pixel existente en esa posición. `opacity` es un `double` de `0.0` a `1.0`. Por ejemplo, `0.0` no cambiará el pixel, y `1.0` ploteará el color dado. Cualquier cosa entre estos dos valores resultará en una combinación de los dos colores.

```
void plot_blend(   int x, int y, double opacity,
                   int red, int green, int blue);
void plot_blend(   int x, int y, double opacity,
                   double red, double green,   double blue);
```

Funciones con Ploteo Combinado

Todas estas funciones son idénticas a sus tipos no-combinadas. Requieren un argumento extra, la opacidad, que es un double que va desde 0.0 a 1.0 y representa cuanto del valor del pixel original se retiene al plotear el nuevo pixel. En otras palabras, si opacidad es 0.7, luego de plotear, el nuevo pixel será 30% del color original y 70% del nuevo color, sea lo que haya sido. Como siempre, cada función esta disponible en versiones `int` y `double`.

```
void plotHSV_blend(    int x, int y, double opacity,
                       double hue, double saturation, double value);
void plotHSV_blend(    int x, int y, double opacity,
                       int hue, int saturation, int value);

void line_blend(   int xfrom, int yfrom, int xto, int yto,
                   double opacity, int red, int green,int  blue);
void line_blend(   int xfrom, int yfrom, int xto, int yto,
                   double opacity, double red, double green,double  blue);

void square_blend( int xfrom, int yfrom, int xto, int yto,
                   double opacity, int red, int green,int  blue);
void square_blend( int xfrom, int yfrom, int xto, int yto,
                   double opacity, double red, double green, double  blue);

void filledsquare_blend(    int xfrom, int yfrom, int xto, int yto,
                            double opacity,
                            int red, int green,int  blue);
```

```
void filledsquare_blend(    int xfrom, int yfrom, int xto, int yto,
                            double opacity,
                            double red, double green,double  blue);

void circle_blend( int xcentre, int ycentre, int radius,
                   double opacity, int red, int green, int blue);
void circle_blend( int xcentre, int ycentre, int radius,
                   double opacity, double red, double green, double blue);

void filledcircle_blend(    int xcentre, int ycentre, int radius,
                            double opacity,
                            int red, int green, int blue);
void filledcircle_blend(    int xcentre, int ycentre, int radius,
                            double opacity,
                            double red, double green, double blue);

void bezier_blend( int startPtX, int startPtY,
                   int startControlX, int startControlY,
                   int endPtX, int endPtY,
                   int endControlX, int endControlY,
                   double opacity,
                   double red, double green, double blue);

void bezier_blend( int startPtX, int startPtY,
                   int startControlX, int startControlY,
                   int endPtX, int endPtY,
                   int endControlX, int endControlY,
                   double opacity,
                   int red, int green, int blue);

void plot_text_blend(  char * face_path, int fontsize,
                       int x_start, int y_start, double angle, char * text,
                       double opacity,
                       double red, double green, double blue);
void plot_text_blend(  char * face_path, int fontsize,
                       int x_start, int y_start, double angle, char * text,
                       double opacity,
                       int red, int green, int blue);

void plot_text_utf8_blend(char * face_path, int fontsize,
                          int x_start, int y_start, double angle, char * text,
                          double opacity,
                          double red, double green, double blue);
void plot_text_utf8_blend(char * face_path, int fontsize,
                          int x_start, int y_start, double angle, char * text,
                          double opacity,
                          int red, int green, int blue);

void boundary_fill_blend(int xstart, int ystart,
         double opacity,
         double  boundary_red,  double  boundary_green,  double  boundary_blue,
         double fill_red, double fill_green, double fill_blue);
```

```
void boundary_fill_blend(int xstart, int ystart,
         double opacity,
         int boundary_red, int boundary_green, int boundary_blue,
         int fill_red, int fill_green, int fill_blue);

void flood_fill_blend(  int xstart, int ystart, double opacity,
                        double fill_red, double fill_green, double fill_blue);
void flood_fill_blend(  int xstart, int ystart, double opacity,
                        int fill_red, int fill_green, int fill_blue);

void polygon_blend(int * points, int number_of_points, double opacity,
                        double red, double green, double blue);

void polygon_blend(int * points, int number_of_points, double opacity,
                        int red, int green, int blue);
void plotCMYK_blend(int x, int y, double opacity,
                  double cyan, double magenta, double yellow, double black);
void plotCMYK_blend(int x, int y, double opacity,
                  int cyan, int magenta, int yellow, int black);
```

Plotear HSV

Igual que el anterior, pero con esta función el color de un pixel en la posición `(x, y)` puede ser cambiado en el espacio de colores de Hue, Saturation, Value. Los coeficientes de color van desde `0` a `65535` si son de tipo `int`, o desde `0.0` hasta `1.0` si son de tipo `double`.

```
void plotHSV(int x, int y, double hue, double saturation, double value);
void plotHSV(int x, int y, int hue, int saturation, int value);
```

Plotear CMYK

Plotea un pixel en el espacio de colores Cyan, Magenta, Amarillo, Negro. Nota que este espacio de colores tiene pérdidas, es decir, no puede reproducir todos los colores que puede reproducir RGB. La diferencia, sin embargo, es casi indistinguible. El algoritmo usado es estándar. Los componentes de color son double desde `0.0` hasta `1.0` o int desde `0` hasta `65535`.

```
void plotCMYK(     int x, int y,
                   double cyan, double magenta, double yellow, double black);
void plotCMYK(     int x, int y,
                   int cyan, int magenta, int yellow, int black);
```

Borrar todo

Toda la imagen se pinta de negro.

```
void clear(void);
```

Invertir

Invierte la imagen en el espacio de colores RGB.

```
void invert(void);
```

Rellenar Con Borde

Todos los pixeles adyacentes al pixel de partida se llenarán con el color de relleno, hasta que se encuentren pixeles del color de borde. Por ejemplo, llamando a `boundary_fill()` con rojo para el color de borde, en un pixel en alguna parte en el interior de un círculo rojo, llenará el círculo con el color de llenado. Si, por el contrario, el círculo no es del color de borde, el resto de la imagen será pintada. Los coeficientes de color son double desde `0.0` a `1.0` o int desde `0` hasta `65535`.

```
void boundary_fill(int xstart, int ystart,
         double boundary_red, double boundary_green, double boundary_blue,
         double fill_red, double fill_green, double fill_blue);
void boundary_fill( int xstart, int ystart,
                    int boundary_red, int boundary_green, int boundary_blue,
                    int fill_red, int fill_green, int fill_blue) ;
```

Rellenar

Todos los pixeles adyacentes al pixel de partida se llenarán con el color de llenado, si son del mismo color que el pixel de partida. Por ejemplo, llamando a `flood_fill()` alguna parte en el interior de un rectángulo sólido azul lo coloreará entero del color de relleno. Los componentes de color son double desde `0.0` hasta `1.0` o int desde `0` hasta `65535`.

```
void flood_fill(   int xstart, int ystart,
                   double fill_red, double fill_green, double fill_blue);
void flood_fill(   int xstart, int ystart,
                   int fill_red, int fill_green, int fill_blue) ;
```

Laplaciano

Esta función aplica el laplaciano discreto a la imagen, multiplicando por un factor constante. El kernel usado en este caso es:

```
1.0  1.0  1.0
1.0 -8.0  1.0
1.0  1.0  1.0
```

Básicamente, esto funciona como un detector de bordes. Al pixel presente se le asigna una suma de sus vecinos, multiplicando por el correspondiente elemento del kernel. Por ejemplo, imagina un pixel y sus 8 vecinos:

```
1.0      1.0      1.0      0.0      0.0
1.0      1.0    ->1.0<-    0.0      0.0
1.0      1.0      1.0      0.0      0.0
```

Esto representa un borde entre negro y blanco, donde el negro está a la derecha. Aplicando el laplaciano a este pixel nos da:

```
1.0*1.0   +  1.0*1.0    +  0.0*1.0   +
1.0*1.0   +  1.0*-8.0   +  0.0*1.0   +
1.0*1.0   +  1.0*1.0    +  0.0*1.0   =  -3.0
```

Aplicando el laplaciano al pixel inmediatamente a la derecha del que acabamos de considerar, obtenemos una suma de `3.0`. Es decir, al pasar sobre un borde, obtenemos un valor grande para el pixel adyacente al borde. Dado que PNGwriter limita los componentes de color si está´n fuera de rango, y el resultado del laplaciano puede ser negativo, se incluye un offset y un factor de escala. Est podría ser útil para que las cosas queden dentro de rango o para resltar más detalle en la detección de bordes. El valor final del pixel está dado por:

```
final value =  laplacian(original pixel)*k + offset
```

Sugerencia: Prueba con un valor de 1.0 para k para partir, y luego experimenta con otros valores. **Sugerencia**: Sería difícil anticipar todos los usos posibles de esta función, así que por favor revisa el código fuente e implementa tu propia versión si ésta no te sirve.

```
void laplacian(double k, double offset);
```

. . . . . Leer . . . . . .

Read

Con esta función averiguamos el color del pixel `(x, y).` Si "color" es `1`, devolverá el coeficiente de color rojo, si es `2`, el verde, si es `3`, el azul. El valor devuelto será de tipo `int` y estará entre `0` y `65535`.

```
int read(int x, int y, int color);
```

Read, Promedio

Lo mismo que el anterior, pero el promedio de los tres coeficientes de color es devuelto.

```
int read(int x, int y);
```

dRead

Igual que Read, pero el valor devuelto será de tipo `double` y estará entre `0.0` y `1.0`.

```
double dread(int x, int y, int color);
```

dRead, Promedio

Lo mismo que el anterior, pero el promedio de los tres coeficientes de color es devuelto.

```
double dread(int x, int y);
```

Interpolación Bilineal de la Imagen

Dado una coordenada de punto flotante (x de `0.0` a `width`, y de `0.0` a `height`), esta función devolverá la intensidad de color interpolada, determinando el color por colour (donde rojo = `1`, verde = `2` y azul = `3`). `bilinear_interpolate_read()` devuelve un `int` de `0` a `65535`, y `bilinear_interpolate_dread()` devuelve un `double` de `0.0` a `1.0`. **Sugerencia**: Es especialmente útil para agrandar una imagen.

```
int bilinear_interpolation_read(double x, double y, int colour);
double bilinear_interpolation_dread(double x, double y, int colour);
```

Read HSV

Con esta función averiguamos el color del pixel `(x, y)`, pero en el espacio de colores de HSV. Si "color" es `1`, devolverá el coeficiente de color Hue, si es `2`, el Saturation, si es `3`, el Value. El valor devuelto será de tipo `int` y estará entre `0` y `65535`. **Importante**: Si intentas leer el Hue de un pixel que es un tono de gris, entonces el valor devuelto no tendrá sentido o será un NaN. Esta es simplemente la manera en que funciona el algoritmo RGB -> HSV: el Hue de un tono de gris no está definido. Antes de usar readHSV(), quizás quieras ver si el pixel es gris o no.

```
int readHSV(int x, int y, int color);
```

dRead HSV

Igual que el anterior, pero el valor devuelto será de tipo `double` y estará entre `0.0` y `1.0`.

```
double dreadHSV(int x, int y, int color);
```

Leer CMYK, versión double

Averigua el color de un pixel en el espacio de colores Cyan, Magenta, Amarillo, Negro. Si 'colour' es `1`, se devolverá el componente Cyan del pixel. Si 'colour' es `2`, se devolverá el componente Magenta, y así, hasta `4`. El valor devuelto será un double de `0.0` hasta `1.0`.

```
double dreadCMYK(int x, int y, int colour);
```

Leer CMYK

Igual que el anterior, pero los componentes que se devuelven son int desde `0` hasta `65535`.

```
int readCMYK(int x, int y, int colour);
```

. . . . . Figuras . . . . . .

Estas funciones dibujan figuras básicas. Disponibles en versiones `int` y `double`.

Línea

Las funciones line usan el algoritmo rápido de Bresenham.

```
void line(    int xdesde, int ydesde,
              int xhasta, int yhasta,
              int rojo, int verde,int azul);
void line(    int xdesde, int ydesde,
              int xhasta, int yhasta,
              double rojo, double verde,double azul);
```

Triangulo

Dibuja el triángulo especificado por los tres pares de puntos.

```
void triangle(     int x1, int y1, int x2, int y2, int x3, int y3,
                   int red, int green, int blue);
void triangle(     int x1, int y1, int x2, int y2, int x3, int y3,
                   double red, double green, double blue);
```

Triángulo Llenado, y Triángulo Llenado combinado

Dibuja el triángulo dado por los tres pares de coordenadas, en el color especificado por los coeficientes de color. filledtriangle_blend() hace lo mismo, pero combinado con el fondo (ver descripción para las funciones combinadas) Los coeficientes de color son `double`s que van desde 0.0 hasta 1.0 o `int`s que van desde `0` hasta `65535`.

```
void filledtriangle(    int x1,int y1,int x2,int y2,int x3,int y3,
                        int red, int green, int blue);
void filledtriangle(    int x1,int y1,int x2,int y2,int x3,int y3,
                        double red, double green, double blue);
void filledtriangle_blend(  int x1,int y1,int x2,int y2,int x3,int y3,
                    double opacity, int red, int green, int blue);
void filledtriangle_blend(  int x1,int y1,int x2,int y2,int x3,int y3,
                    double opacity, double red, double green, double blue);
```

Cuadrado

A pesar de su nombre, estas funciones dibujan rectángulos. Filledsquare dibuja un cuadrado llenado.

```
void square(  int xdesde, int ydesde,
              int xhasta, int yhasta,
              int rojo, int verde,int azul);
void square(  int xdesde, int ydesde,
              int xhasta, int yhasta,
              double rojo, double verde,double azul);
void filledsquare( int xdesde, int ydesde,
                   int xhasta, int yhasta,
                   int rojo, int verde,int azul);
void filledsquare( int xdesde, int ydesde,
                   int xhasta, int yhasta,
                   double rojo, double verde,double azul);
```

Círculo

Las funciones de círculo usan un algoritmo rápido de operaciones con números enteros. Las funciones de círculo llenado usan la función sqrt().

```
void circle(  int xcentro, int ycentro, int radio,
              int rojo, int verde, int azul);
void circle(  int xcentro, int ycentro, int radio,
              double rojo, double verde, double azul);
```

```
void filledcircle( int xcentro, int ycentro, int radio,
                   int rojo, int verde, int azul);
void filledcircle( int xcentro, int ycentro, int radio,
                   double rojo, double verde, double azul);
```

Curva Bezier

(llamado así en honor del frances Pierre Bézier de Regie Renault) Una colección de fórmulas para describir líneas y superficies curvas, usados por primera vez en 1972 para modelar las lineas curvas de automóviles. (de The Free Online Dictionary of Computing). Ver http://www.moshplant.com/direct-or/bezier/ para una de las muchas descripciones disponibles acerca de las curvas de bezier. Hay cuatro puntos usados para definir la curva: los dos extremos se llaman los puntos de anclaje, mientras que los otros puntos, que describen la curvatura, se llaman puntos de control. Moviendo los puntos de control podemos modificar la forma de la curva.

```
void bezier(  int startPtX, int startPtY,
              int startControlX, int startControlY,
              int endPtX, int endPtY,
              int endControlX, int endControlY,
              double red, double green, double blue);

void bezier(  int startPtX, int startPtY,
              int startControlX, int startControlY,
              int endPtX, int endPtY,
              int endControlX, int endControlY,
              int red, int green, int blue);
```

Polígono

Esta función toma un arreglo de valores enteros representando las corrdenadas de los vértices de un polígono. Nota que si quieres un polígono cerrado, debes repetir las coordenadas del primer punto al final del arreglo. También debes especificar cuantos puntos contiene el arreglo. Por ejemplo, si quieres plotear un triángulo, entonces el arreglo contendrá `6` elementos, y el número de los puntos es `3`. Ten mucho cuidado con esto: si especificas el numero equivocado de puntos, tu programa tirará un segfault o ploteará vértices en coordenadas insensatas. Los componentes de color son double desde `0.0` hasta `1.0` o int desde `0` hasta `65535`.

```
void polygon( int * points, int number_of_points,
              double red, double green, double blue);
void polygon( int * points, int number_of_points,
              int red, int green, int blue);
```

Flecha, Flecha Sólida

Plotea una flecha desde `(x1, y1)` hasta `(x2, y2)` con la cabeza de la flecha en el segundo punto, dado el tamaño en pixeles y el ángulo de la cabeza en radiantes. La flecha consiste en una línea principal y dos líneas secundarias originando del segundo punto. La flecha sólida es igual, pero la cabeza es un triángulo sólido. **Sugerencia:** Un ángulo de `10` a `30` grados (`0.1745` a `0.5236` radianes) se ve bien.

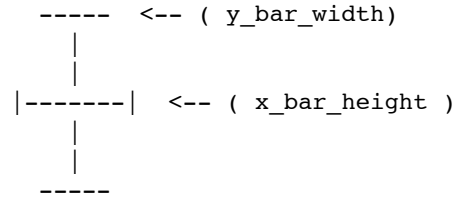
```
void arrow( int x1, int y1, int x2, int y2,
         int size, double head_angle, double red, double green, double blue);
```

```
void arrow( int x1, int y1, int x2, int y2,
         int size, double head_angle, int red, int green, int blue);

void filledarrow(  int x1, int y1, int x2, int y2,
                   int size, double head_angle,
                   double red, double green, double blue);
void filledarrow(  int x1, int y1, int x2, int y2,
                   int size, double head_angle,
                   int red, int green, int blue);
```

Cruz, Cruz de Malta

Plotea una cruz simple en `(x, y)`, con la altura y ancho especificados, y en el color dado. Cruz de malta plotea una cruz, al igual que el caso anterior, pero le agrega barras al final de cada extremidad de la cruz. El tamaño de estas barras está dado por `x_bar_height` y `y_bar_width`. La cruz se verá algo así:




```
void cross(   int x, int y, int xwidth, int yheight,
              double red, double green, double blue);
void cross(   int x, int y, int xwidth, int yheight,
              int red, int green, int blue);

void maltesecross( int x, int y, int xwidth, int yheight,
                   int x_bar_height, int y_bar_width,
                   double red, double green, double blue);
void maltesecross( int x, int y, int xwidth, int yheight,
                   int x_bar_height, int y_bar_width,
                   int red, int green, int blue);
```

Diamante, Diamante Sólido

Plotea una figura de diamante, dada la posición (`x`, `y`), el ancho y altura, y el color. Diamante sólido hace lo mismo pero llenado.

```
void diamond( int x, int y, int width, int height,
              int red, int green, int blue);
void diamond( int x, int y, int width, int height,
              double red, double green, double blue);

void filleddiamond(    int x, int y, int width, int height,
                       int red, int green, int blue);
void filleddiamond(    int x, int y, int width, int height,
                       double red, double green, double blue);
```

. . . . . Texto . . . . . .

Plotear Texto

Usa la librería FreeType2 para plotear texto en la imagen. `face_path` es el path donde se encuentra un archivo de font TrueType (`.ttf`). (FreeType2 también puede usar otros tipos). `fontsize` especifica el tamaño aproximado del font dibujado, en pixeles. `x_start` e `y_start` especifican la ubicación de la esquina inferior izquierda del string de texto. `angle` es el ángulo que el texto formará con la horizontal, en radiantes. `text` es el texto por dibujar. Las coordenadas de color pueden ser doubles de `0.0` a `1.0` o `ints` de `0` a `65535`. **Sugerencia**: PNGwriter instala algunos fonts en `/usr/local/share/pngwriter/fonts` para que puedas comenzar. Otra sugerencia: recuerda de agregar `-DNO_FREETYPE` a tus flags de compilación si PNGwriter fue compilado sin soporte de FreeType.

```
void plot_text(    char * face_path, int fontsize,
                   int x_start, int y_start,double angle,
                   char * text,
                   double red, double green, double blue);
void plot_text(    char * face_path, int fontsize,
                   int x_start, int y_start,   double angle,
                   char * text,
                   int red, int green, int blue);
```

Plotear Texto UTF-8

Igual que el anterior, pero el texto que se ploteará está codificado en UTF-8. Por qué querrías algo así? Para poder plotear caracteres con acentos, y todos los caracteres disponibles en un font TrueType, como por ejemplo para plotear texto en Japonés, Chino y otros idiomas no restringidos al espacio ASCII estándar de 128 caracteres. **Sugerencia**: La manera más rápida de convertir un string a UTF-8 es escribirlo en un editor de texto adecuado y luego guardarlo como un archivo con codificación UTF-8, el cual posteriormente podrá ser leído en modo binario por tu programa.

```
void plot_text_utf8(    char * face_path, int fontsize,
                        int x_start, int y_start, double angle,
                        char * text,
                        double red, double green, double blue);
void plot_text_utf8(    char * face_path, int fontsize,
                        int x_start, int y_start, double angle,
                        char * text, int red, int green, int blue);
```

Calcular el Ancho del Texto

Da el ancho aproximado, en pixeles, del texto *sin rotar* dado. Se calcula sumando el ancho y kerning de cada letra, como lo especifica el archivo TTF. Nota que esto no te dará la posición del pixel más lejano, pero sí una idea de qué área ocupará el texto. **Sugerencia:** El texto, si se plotea sin rotar, cabrá aproximadamente dentro de una caja con su esquina inferior izquierda en `(x_start, y_start)` y superior derecha en `(x_start + width, y_start + size)`, donde el ancho se obtiene de `get_text_width()` y la la altura se obtiene del tamaño del texto a ser ploteado. **Sugerencia:** El texto que se plotea en la posición `(x_start, y_start)`, rotado con un ángulo '`th`', y de un tamaño '`size`', cuyo ancho es '`width`', cabrá aproximadamente en un rectángulo con esquinas en:

```
     (x_start, y_start)
     (x_start + width*cos(th), y_start + width*sin(th))
     (x_start + width*cos(th) - size*sin(th),y_start+ width*sin(th)+size*cos(th))
     (x_start - size*sin(th), y_start + size*cos(th))

int get_text_width(char * face_path, int fontsize,  char * text);
int get_text_width_utf8(char * face_path, int fontsize, char * text);
```

Tamaño de la Imagen

Escala Proporcional

La imagen se re-escalada usando interpolación bilineal. Si k es mayor que 1.0, la imagen se agrandará. Si k es menor que 1.0, la imagen se achicará. No se permiten valores negativos o nulos para k. La imagen será re-escalada y el contenido previo será reemplazado por el resultado. Sugerencia: usa getheight() y getwidth() para averiguar el ancho y la altura de la imagen escalada. **Nota**: Después de re-escalar, todas las imágenes tendrán bit depth de 16, aún si la imagen original tenía un bit depth de 8.

```
void scale_k(double k);
```

Escala No-Proporcional

La imagen será re-escalada usando interpolación bilineal, con factores de escala horizontal y vertical diferentes.

```
void scale_kxky(double kx, double ky);
```

Escala dado el ancho y altura finales

La imagen será re-escalada de tal manera de obtener una imagen con el ancho y altura dados. **Sugerencia**: Si quieres que la imagen quede proporcional, podría ser mejor usar scale_k().

```
void scale_wh(int finalwidth, int finalheight);
```

Resize Image

Cambia el tamaño de la instancia de PNGwriter. **Nota**: toda la imagen se resetea a negro (esto es un cambio de tamaño, no un re-escalamiento).

```
void resize(int width, int height);
```

Funciones de Archivo y PNG

Leer Desde Archivo

Abre el archivo PNG ya existente y lo incorpora a su almacenamiento. **Nota**: Sólo se pueden abrir de esta manera imágenes PNG de bit depth 8 o 16.

```
void readfromfile(char * name);
void readfromfile(const char * name);
```

Cerrar
Cerrar la instancia de la clase, y escribir la imagen al disco.

```
void close(void);
```

Cambiar el nombre
Para cambiar el nombre del archivo una vez que la instancia de PNGwriter ha sido creada. Si el argumento es un `long unsigned int`, por ejemplo `77`, el nombre del archivo será `000000077.png`.

```
void pngwriter_rename(char * nuevonombre);
void pngwriter_rename(const char * nuevonombre);
void pngwriter_rename(long unsigned int index);
```

Averiguar Altura
Cuando abres un PNG ya existente, puedes averiguar su altura con esta función.

```
int getheight(void);
```

Averiguar Ancho
Cuando abres un PNG ya existente, puedes averiguar su ancho con esta función.

```
int getwidth(void);
```

Escribir PNG
Escribe el PNG al disco. Después de esto, todavía puedes seguir alterando la instancia de PNGwriter.
```
void write_png(void);
```

Elegir Nivel de Compresión
Elegir el nivel de compresion que será usada para la imagen. `-1` selecciona la compresión por defecto, `0` es ninguna, `9` es la mejor compresión.

```
void setcompressionlevel(int nivel);
```

Averiguar Bit Depth
Cuando abres un PNG ya existente, puedes averiguar su profundidad de bits con esta función.

```
int getbitdepth(void);
```

Averiguar Colour Type

Cuando abres un PNG ya existente, puedes averiguar su color type con esta función.

```
int getcolortype(void);
```

Cambiar Gamma

Cambiar el gamma de la imagen (filegamma). El valor prefijado de 0.5 debería estar bien.

```
void setgamma(double gamma);
```

Averiguar Gamma

Averigua el coeficiente de gamma de la imagen. Esto es experimental.

```
double getgamma(void);
```

Cambiar Texto

Cambia el texto contenido en el header del archivo PNG. Si esta función no se llama, se usara el texto por defecto.

```
void settext( char * title, char * author,
              char * description, char * software);
void settext( const char * title, const char * author,
              const char * description, const char * software);
```