Ejercicio obligatorio 4

Fecha de entrega: Domingo 5 de junio

Introducción

El archivo

En nuestro videojuego una figura es la representación gráfica de una entidad que puede ser un personaje, un ícono, parte de un nivel, que luego se dibujará en la pantalla de forma visual.

El juego codifica sus figuras en un archivo binario. Este archivo contiene una sucesión de figuras, donde cada figura se compone de un par de parámetros sobre la misma y luego de una sucesión de polilíneas.

Un archivo de figuras tendrá entonces este aspecto:

```
+----------+----------+----------+---------+----------+
| Figura 1 | Figura 2 | Figura 3 |   ...   | Figura N |
+----------+----------+----------+---------+----------+
```

es decir, una sucesión de un largo desconocido de figuras.

Luego cada figura tendrá este aspecto:

```
+------------+---------+---------+-------+---------+
| Encabezado | Polil 1 | Polil 2 |  ...  | Polil N |
+------------+---------+---------+-------+---------+
```

Donde el encabezado es una secuencia de:

Nombre:

20 caracteres que representan el nombre de la figura, finalizados con `'\0'`.

Características:

Un byte que codifica el tipo y las propiedades según el siguiente esquema:

```
MSB                                 LSB
+----+----+----+----+----+----+----+----+
| -- | IN | -- | -- | T2 | T1 | T0 | -- |
+----+----+----+----+----+----+----+----+
```

donde el paquete `T` representa el tipo y el bit `IN` indica si es una figura infinita.

El tipo puede ser uno de estos valores:

Valor	Tipo
0	Icono
1	Nivel
2	Sprite

(Y esta lista puede crecer, por eso hay 3 bits reservados.)

Número de polilíneas:

Un entero sin signo de 16 bits que codifica la cantidad de polilíneas que vienen a continuación.

Las polilíneas

Una polilínea se representa en un archivo como una sucesión:

```
+------------+---------+---------+-----+---------+
| Encabezado | Punto 1 | Punto 2 | ... | Punto N |
+------------+---------+---------+-----+---------+
```

donde el encabezado es un número de 16 bits que representa lo siguiente:

```
MSB                                                                   LSB
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
| R  | G  | B  | -- | -- | -- | N9 | N8 | N7 | N6 | N5 | N4 | N3 | N2 | N1 | N0 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
```

donde los bits R, G, B representan la presencia de esa componente de color y el paquete de los 10 bits N representan la cantidad de puntos.

Luego cada punto es una secuencia de pares de tipo `float`.

Tipos

Color

Los colores de nuestro juego son de 3 bits, es decir, se permiten 8 colores posibles. Vamos a almacenarlo en un byte

```
typedef uint8_t color_t;
```

Implementar la primitiva `color_t color_crear(bool r, bool g, bool b);` que cree un `color_t` en base a los valores de las compontentes de RGB.

Implementar la primitiva `void color_a_rgb(color_t c, uint8_t *r, uint8_t *g, uint8_t *b);` que convierta un `color_t` a las componentes de una representación RGB en 24 bits, donde cada una de las componentes valdrá 0 o 255 dependiendo de si ese color estaraba presente o no.

Tipo de figura

Definir un enumerativo `figura_tipo_t` que contenga etiquetas para los tipos de figura.

Programar una función `const char* figura_tipo_a_cadena(figura_tipo_t figura);` que devuelva una cadena que represente ese tipo de figura. La misma debe ser implementada con tablas de búsqueda.

Polilínea

Se provee una definición de `polilinea_t` similar a la del EJ3 pero que no implementa ninguna lógica. Esta polilínea en el futuro puede ser reemplazada por la implementación final de dicho ejercicio.

A esta polilínea del EJ3 se le agregó un `color_t` con su respectivo getter y setter.

La misma es:

```c
struct polilinea;
typedef struct polilinea polilinea_t;

struct polilinea {
};

polilinea_t *polilinea_crear_vacia(size_t n) {
    printf("POLILINEA N=%zd\n", n);
    static polilinea_t x;
    return &x;
}

void polilinea_destruir(polilinea_t *polilinea) {}

bool polilinea_setear_punto(polilinea_t *polilinea, size_t pos, float x, float y) {
    printf("POLILINEA[%zd] = (%.2f, %.2f)\n", pos, x, y);
    return true;
}

bool polilinea_setear_color(polilinea_t *polilinea, color_t color) {
    uint8_t r, g, b;
    color_a_rgb(color, &r, &g, &b);
    printf("POLILINEA COLOR: (%d, %d, %d)\n", r, g, b);
    return true;
}
```

Lectura

Desarrollar una función `bool leer_encabezado_figura(FILE *f, char nombre[], figura_tipo_t *tipo, bool *infinito, size_t *cantidad_polilineas);` que reciba un archivo `f` e intente leer el encabezado de una figura. Debe devolver el `nombre`, el `tipo`, si es de tipo `infinito` y la cantidad de polilíneas asociadas. Si la lectura es correcta debe devolver `true`.

Desarrollar una funcion `polilinea_t *leer_polilinea(FILE *f)` que lea y devuelva una polilínea. Si hay alguna falla debe devolver `NULL`. Utilizar las primitivas de polilínea provistas.

Aplicación

Se provee el siguiente `main()`:

```c
int main(int argc, char *argv[]) {
    if(argc != 2) {
        fprintf(stderr, "Uso: %s <archivo>\n", argv[0]);
        return 1;
    }

    FILE *f = fopen(argv[1], "rb");
    if(f == NULL) {
        fprintf(stderr, "No pudo abrirse \"%s\"\n", argv[1]);
        return 1;
    }

    int figura = 0;
    while(1) {
        figura++;

        char nombre[20];
        bool infinito;
        figura_tipo_t tipo;
        size_t cantidad_polilineas;

        if(! leer_encabezado_figura(f, nombre, &tipo, &infinito, &cantidad_polilineas))
            break;

        printf("FIGURA \"%s\", TIPO: %s, INFINITO: %d, POLILINEAS: %zd\n", nombre,
figura_tipo_a_cadena(tipo), infinito, cantidad_polilineas);

        for(size_t i = 0; i < cantidad_polilineas; i++) {
            polilinea_t *p = leer_polilinea(f);
            if(p == NULL) {
                fprintf(stderr, "Error en el archivo");
                fclose(f);
                return 1;
            }
            polilinea_destruir(p);
        }
    }

    fclose(f);

    return 0;
}
```

Y además se provee el siguiente juegos de archivos: archivos_20221_ej4.tar.gz de una entrada con su respectiva salida.

Este `main()`, sin alteraciones, debe ser entregado como parte del ejercicio y el mismo debe correr para el archivo de entrada generando la salida pedida.

Nota

Para este trabajo no es necesario modularizar el problema, pero se recomienda hacerlo para practicar modularización y Makefile.

En el caso de modularizar en archivos entregar tanto los archivos como el Makefile que compila el proyecto.

Entrega

Deberá entregarse el código fuente del programa desarrollado o los fuentes y el Makefile en caso de haber modularizado.

El programa debe:

1. Compilar correctamente con los flags:

```
-Wall -Werror -std=c99 -pedantic
```

2. validar la salida contra los ejemplos dados.

La entrega se realiza a través del sistema de entregas.

El ejercicio es de entrega individual.