Lab 04 virtual: Punteros y estructuras de datos

Nivel 3

1. Objetivo

Este laboratorio busca introducir a los estudiantes en el manejo de punteros, memoria y estructuras de datos en el lenguaje C, para así conocer una mayor cantidad de herramientas que pueden ser utilizadas en este lenguaje. Esto permitirá emular la forma de controlar la información de una aplicación que utilizaría protocolos de comunicación en un microcontrolador.

Lo realizado en este laboratorio, le permitiría ser reutilizado (con las modificaciones pertinentes) en aplicaciones que requieren comunicar diversos dispositivos entre sí.

2. TASK: SEP SIMON GAME

2.1. Introducción

Simon fue un juego originalmente creado en 1978, por Ralph Baer y Howard J. Morrison.[1] Esta conformado por 4 cuadrantes de color rojo, verde, amarillo y azul que repiten distintas series las que deben ser memorizadas y repetidas. El juego concluye cuando el jugador no es capaz de recordar la secuencia y falla al repetirla.

La siguiente es una imagen del juego Simon fabricado por Hasbro [2]:

Figura 1: Simon Game, propiedad de Hasbro. Imagen obtenida de www.amazon.com

[1] https://en.wikipedia.org/wiki/Simon_(game)

[2] https://www.amazon.com/-/es/B7962-Juego-Simon/dp/B01ALHAN7Q

Para hacerse una idea del funcionamiento de este juego, pueden observar la siguiente video https://www.youtube.com/watch?v=1Yqj76Q4jJ4 en el que aparece la explicación de este juego.

2.2. Actividad a realizar

Para esta actividad, deberá recrear este juego utilizando el lenguaje C y su conocimiento en punteros, estructuras de datos y manejo de memoria. Para ayudarlo en esta misión, se el entregará la siguiente interfaz básica que les permitirá visualizar el juego.

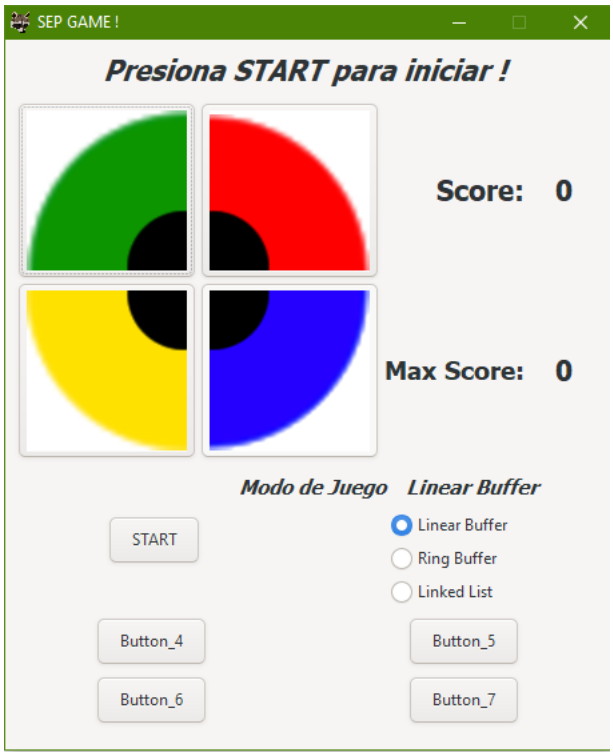

Figura 2: Interfaz Simon SEP GAME

La interfaz se explicará en una sección más adelante, para esta actividad deberán implementar 3 modos de juegos con distintos buffers, los que serán utilizados para:

1. Guardar la secuencia a memorizar.
2. Guardar la secuencia ingresada por el usuario y chequearla con la ya guardada.

2.2.1. Linear Buffer 20 %

El modo de juego de Linear Buffer es el modo de juego más sencillo. Deberá utilizar esta estructura de datos para almacenar la secuencia creada. Dada las características de este tipo de estructura de datos, hay un tope de información que puede guardar, por lo que si el jugador logra llegar a este tope habrá ganado la partida y podrá volver a jugar nuevamente.

Debe cumplir con las siguientes características:

- El tamaño del buffer debe ser un parámetro, de forma que sea fácilmente modificable antes de compilar.
- No es necesario que el tamaño sea posible cambiarlo una vez ya se compiló el programa o cuando se está ejecutando el juego.
- La secuencia a memorizar debe ser aleatoria, por lo que tiene que cambiar cada vez que se presiona **START**.
- Se debe guardar el puntaje logrado y mostrarlo en la interfaz.
- El juego termina cuando se alcanza el tamaño máximo del buffer o el jugador pierde al anotar la secuencia.

2.3. Ring Buffer 30 %

Para este modo de juego, se debe utilizar la estructura de datos Ring Buffer. A diferencia del anterior, este permite guardar los últimos n elementos de un arreglo, por lo que la complejidad de este modo será recordar estos últimos n elementos. El juego termina cuando el jugador se equivoca en la secuencia a memorizar.

Debe cumplir las siguientes características:

- El tamaño del Ring Buffer, denominado n, debe ser un parámetro, de forma que sea fácilmente modificable antes de compilar.
- No es necesario que el tamaño sea posible cambiarlo una vez se compiló o mientras se está ejecutando el juego.
- La secuencia a memorizar debe ser aleatoria, por lo que tiene que cambiar cada vez que se presiona **START**.
- Se deben mostrar los último n elementos asignados, por lo que es importante tener buen control de los punteros de lectura y escritura.

- Se debe guardar el puntaje logrado y mostrarse en la interfaz.
- El juego termina cuando el jugador pierde al anotar la secuencia.

2.3.1. Linked List 40 %

En este modo de juego, no hay límite teórico al que se podría llegar, solo está la capacidad de memoria del jugador. Es por esto que deberá utilizar una Linked List para este propósito, debido que no será posible saber cuantas secuencias será capaz de recordar el usuario. (**SPOILER: sus ayudantes y profesor puede que tengan muy buena memoria**).

Debe cumplir las siguientes características:

- El tamaño es variable, por lo que se debe comenzar con un nodo cabeza e ir creciendo a partir de él.
- Se debe cuidar la memoria a utilizar, por lo que no debe olvidar hacer uso de `malloc` y `free`.
- Se debe guardar el puntaje logrado y mostrarse en la interfaz.
- El juego termina cuando el jugador se equivoca en realizar la secuencia.
- No hay límite teórico, por lo que la secuencia debe continuar hasta que el jugador falle.
- La secuencia a memorizar debe ser aleatoria, por lo que tiene que cambiar cada vez que se presiona **START**.

2.4. Características comunes a todos los modos de juego 10 %

Las siguientes características deben ser comunes a todos los modos de juegos.

- El buffer a utilizar solo debe ser seleccionado al momento de iniciar el juego o al terminar (particularmente en los estados `GM_INIT, GM_END` explicados en la sección interfaz).
- Si se trata de cambiar el tipo de buffer mientras se está jugando y/o mostrando la secuencia (estados `GM_SHOW, GM_BUSY`), este proceso debe ser ignorado.
- El puntaje máximo se debe actualizar al terminar el juego. Para hacerlo más sencillo, este valor será común a los 3 buffers, por lo que no es necesario realizar un control en función de qué tipo de buffer se está utilizando.

- El botón START inicializa un nuevo juego independientemente de lo que esté sucediendo. Si el puntaje actual estaba siendo mayor al máximo, este se debe actualizar.
- El puntaje máximo es por sesión, no por el programa. Por lo que siempre comienza en 0. (Si desea modificar esto, entrará en la categoría de bonus).
- La velocidad del juego está establecida en 800 ms. Este es un parámetro fijo. Si desea modificarlo y crear distintos niveles de dificultad, será considerado como bonus.

3. Bonus

La nota máxima de este laboratorio es 6.0. Existen una serie de bonus que permiten lograr un 7.0, la nota no puede superar este umbral.

3.1. Guardar puntajes máximos históricos (3 décimas)

Si es capaz de guardar el puntaje máximo histórico y cargarlo nuevamente cada vez que se inicia la interfaz. Ya sea leyendo un archivo o utilizando algún otro método.

3.2. Niveles de dificultad (4 décimas)

Si es capaz de poner distintos niveles de dificultad que van disminuyendo el tiempo durante el que se mantiene encendida una animación.

3.3. Guardar puntaje máximo por modo de juego (3 décimas)

Al cambiar de modo de juego de Linear Buffer, Ring Buffer o LinkedList, cada uno de estos tendrá su propio puntaje máximo, el que cambiará al seleccionar cada tipo de búffer.

3.4. Añadir efectos de música a la interfaz (5 décimas)

Agregar efectos sonoros al presionar las imágenes o al mostrar la secuencia de valores.

3.5. Propone tu bonus :)

Si tienes alguna idea que te gustaría implementar, puedes sugerirla en el GitHub y se asignarán décimas dependiendo de la dificultad de la misma. El rango será entre 2 a 5

décimas y solo será válido si se tiene una respuesta positiva por los ayudantes o el profesor en el GitHub.

Para esto, habrá una Issue que agrupará todos los bonus propuestos.

4. Lectura Complementaria

Páginas que hablen de C son útiles. La interfaz fue realizada gracias a los tutoriales de

- Programmers Note's https://www.youtube.com/playlist?list=PLaybP4QvyRH1obigtMQwhB2DWhPKg32_T
- How to use GTK in Windows https://www.youtube.com/watch?v=ksBx4C2NeGw
- Gnome Developer, "The Main Event Loop https://developer.gnome.org/glib/stable/glib-The-Main-Event-Loop.html
- Setting up GTK for Windows https://www.gtk.org/docs/installations/windows/
- C - Header Files https://www.tutorialspoint.com/cprogramming/c_header_files.htm
- Memory management https://www.tutorialspoint.com/cprogramming/c_memory_management.htm
- GCC: información y explicaciones sobre las opciones de la línea de comando de GCC, el compilador recomendado para utilizar en este laboratorio

5. Requerimientos mínimos para ejecutar

Para poder realizar la programación de la interfaz y de su código deben realizar los pasos indicados dependiendo de su sistema operativo.

5.1. Windows

Los pasos de instalación son los siguientes:

1. Instalar compilador de gcc MinGW http://www.mingw.org/. Link directo de descarga https://osdn.net/projects/mingw/downloads/68260/mingw-get-setup.exe/

2. Es importante agregar gcc al PATH de windows, el siguiente tutorial puede ser útil en caso de que tenga problemas. https://www.youtube.com/watch?v=sXW2VLrQ3Bs
3. Instalar MSYS2 en https://www.msys2.org/
4. Seguir hasta el paso Nro 3 en https://www.gtk.org/docs/installations/windows/ (Lo que debe ser realizado en MSYS2)

Para ejecutar y compilar se debe abrir una consola de MSYS2 (64 o 32 bit dependiendo de su computador), moverse a la carpeta utilizando el comando `cd`. Por ejemplo:

```
cd /c/users/USUARIO/Documents/SEP/LAB_04
```

Posterior a esto ejecutar el comando que está en el archivo `compile.script`.

```
./compile.script
```

Finalmente, recomiendo abrir el archivo `main-bin.exe` desde el explorador de archivos de Windows, haciendo doble click sobre él. Esto para que pueda ver los prints que la consola arroja, ya que desde MSYS2 podría no ser inmediatos.

5.2. MAC OS

Para proceder a la instalación del compilador requerimos instalar en primer lugar el gestor de paquetes Homebrew mediante el comando (si ya lo tiene instalado debe omitir)

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master
    /install)"
```

Luego aceptamos e instalamos el complemento de Xcode mediante el siguiente comando (omitir si ya se ha hecho):

```
xcode-select --install
```

Este proceso requiere de varias librerías las cuales se instalarán mediante Homebrew de la siguiente forma:

```
brew install pkg-config

brew install gtk+3

brew install glade
```

Esto instalará los paquetes necesarios, luego para realizar la compilación, se debe acceder a la carpeta que contiene el program mediante el comando `cd`, luego ejecutar

```
bash compile.script
```

IMPORTANTE, en el archivo `main.c` hay que comentar la linea que contiene `#include <windows.h>` dado que esto es solo necesario en Windows.

6. Uso de Interfaz Gráfica

6.1. Archivos

La interfaz gráfica se compone de varios elementos que se encuentran separadas en carpetas.

- **glade**, esta carpeta contiene el archivo `main.glade`, que es la interfaz gráfica desarrollada en Glade.
- **res**, esta carpeta contiene las imágenes que permiten mostrar la animación de las secuencias de colores y un icono de mapache.
- **src**, esta carpeta contiene 3 archivos:
  - `main.c` Contiene el código principal, que es el encargado de la interfaz gráfica. Este archivo es el que **menos debería ser modificado**, ya que todo el control de la lógica del juego debe ser realizado en `game_logic.c`.
  - `game_logic.c`, contiene las funciones que controlarán el juego. Aquí es donde se deberían crear los buffers y el control de información del juego.
  - `game_logic.h`, declaración de funciones y de variables del juego.
- **compile.script** contiene las lineas para compilar.

En `game_logic.h`, se encontrará un `struc` de nombre `app_widgets`, el que contiene información de los Widgets y de las variables del juego. Esta debidamente comentado, pero a modo de destacar contiene las variables:

game_state gm_state; Que indica cuál es el estado actual del juego, pudiendo tomar los valores `GM_INIT, GM_BUSY, GM_SHOW, GM_END`, que indican Inicio del juego, Jugador jugando, Mostrando secuencia y juego terminado respectivamente.

game_mode gm_mode; Que indica cuál es el tipo de buffer que se está utilizando, puede tomar los valores `GM_LinBuff, GM_RingBuff, GM_LinkList`.

Las variables anteriores pueden ser usadas para conocer cómo actuar frente a cada una de las situaciones.

6.2. Elementos

En esta sección se mencionarán los puntos importantes de la interfaz gráfica. Es importante mencionar que cada elemento (o widget) tiene un nombre en específico, que realiza alguna acción en particular. En las siguientes secciones se mostrará cuales son sus acciones.

6.2.1. Botones

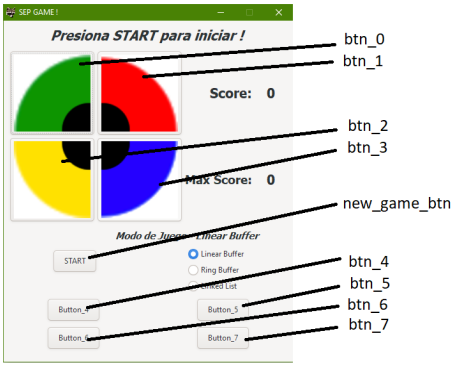

Figura 3: Botones

Cada uno de estos botones esta conectado a la función `on_btn_X_clicked()`, es decir, cada vez que se presiona este botón, la función correspondiente se ejecuta.

- Los botones 0-3, son lo utilizados al jugar, todos llaman a la función `player_round`, la que debe completarse para realizar la lógica del juego.

- El boton `new_game` inicializa un nuevo juego. Además resetea todas las variables.
- Los botones 4-7 se encuentran liberados para el uso que estime conveniente, pueden ser útiles para debuggear.

6.2.2. Labels

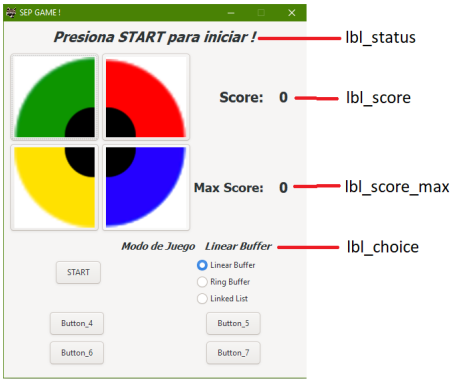

Figura 4: Labels

Los labels permiten mostrar información e ir cambiandolo al ejecutar acciones. Para saber como hacerlo, puede consultar la función `set_text_end_game(app_widgets *app_wdgts)`

6.2.3. Radio Button

Figura 5: Radio

Cada uno de estos botones esta conectado a la función `on_rb_X_toggled()`, es decir, cada vez que se presiona este botón, la función correspondiente se ejecuta. Cabe señalar que pertenecen a un grupo, por lo que nunca pueden haber 2 accionados al mismo tiempo. En el código entregado, al presionar cada uno de estos botones se actualiza la variable `gm_mode`.

6.3. Timers

El programa tiene ejecutnado 2 Timers de forma simúltanea:

- **TIMER 1**
  `gint timer_1 = g_timeout_add_full(G_PRIORITY_HIGH, 800, (GSourceFunc) timer_handler_1, widgets, NULL);`

  Este se activa cada 800 ms con prioridad alta. Llama a la función `show_mem_function`, que es la que debería ser encargada de mostrar la secuencia a memorizar.

- **TIMER 2**
  `gint timer_2 = g_timeout_add(10, (GSourceFunc)timer_handler_2, widgets);`

  Este timer se activa cada 10 ms, permite contabilizar y realizar la animación de "presionar un botón". Que disminuye un contador hasta llegar a 0 y cambiar la imagen.

7. Fecha de entrega

Se tiene hasta el día domingo 7 de junio a las 23:59 hrs. Deberá subir su **código completo y ordenado** a su repositorio de GitHub junto con un informe en formato PDF de no más de 2 páginas aclarando su código, indicando que realizó y que no.

Su código debe estar debidamente comentado, ya que no habrá revisión presencial y esta será la única forma de presentar.

En el README debe indicar qué sistema operativo utilizó. Piensen que el archivo a entregar debiera funcionar al hacer click sobre el archivo generado.