

Ejercicio obligatorio 3

Fecha de entrega: Domingo 5 de mayo

📌 Nota

Nunca es buena idea empezar por un ejercicio integrador antes de tener practicados los temas que el trabajo integra.

Se sugiere antes de desarrollar este trabajo resolver, al menos, los siguientes ejercicios de la guía de Memoria dinámica:

- Ejercicio 12 (combinar vectores).
- Ejercicio 14.a (matriz identidad).
- Ejercicio 16 con el 1.b y 1.c de la guía de estructuras (dirección y persona).

Códigos de escape ANSI

En C tanto la entrada como la salida se comportan como un flujo continuo de datos. La entrada en C además es con eco, es decir, cuando el usuario presiona una tecla esta tecla se muestra en la pantalla, asimismo esta entrada es con buffer, hace falta que el usuario presione Enter para que la salida esté disponible.

Este comportamiento es una limitación para interactuar en una terminal e imprimir salida con formato y para crear interfases de usuario.

Para mejorar esta interacción, las terminales implementan la posibilidad de cambiar su comportamiento al recibir determinadas secuencias de códigos. El estándar ANSI X3.64 describe secuencias de comandos y sus comportamientos. Estos comandos controlan la posición del cursor, el color de la terminal, entre otras cosas.

Las secuencias ANSI comienzan con un caracter Escape y un corchete, luego viene el comando.

Por ejemplo, la secuencia de activación de ANSI ESC[seguida de la secuencia `2K` borra la línea actual. El carácter Escape es el carácter 27, 033 o 0x1b según la base que utilicemos. Si imprimiéramos:

```
printf("\nHo!a\033[2KChau");
```

mostraría por la pantalla `" Chau"` en una línea nueva. ¿Dónde está `"Ho!a"`?, se borró, ¿y por qué hay 4 espacios al comienzo?, porque el cursor quedó en el mismo lugar en el que estaba después de imprimir la `'a'`.

Si quisiéramos además mover el cursor en la línea podemos usar la secuencia `nG`, donde `n` es la columna en la cual queremos ponerlo. Entonces:

```
printf("\nHo!a\033[2K\033[1GChau");
```

mostrará `"Chau"`, dado que borramos la línea y movimos el cursor al comienzo.

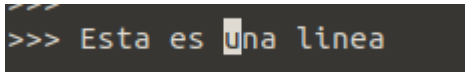
Para imprimir en colores tenemos las siguientes secuencias:

- `0;0m` Resetea al color por omisión,
- `1;31m` Rojo,
- `1;32m` Verde,
- `1;33m` Amarillo,
- `1;34m` Azul,
- `1;35m` Magenta,
- `1;36m` Cyan,
- `1;37m` Blanco.

Cuando vamos a manipular el cursor, son prácticos dos comandos: `S` Guarda la posición actual del cursor, `u` Recupera la posición que se guardó antes.

Edición de línea de terminal

Supongamos una terminal donde el usuario ingresó la cadena `"Esta es una línea"` y luego presionó 9 veces la tecla izquierda para mover el cursor para que quede debajo de la letra `'u'`. Esto se vería así:

A screenshot of a terminal window with a dark background. The prompt is three greater-than signs (>>>). The text "Esta es una línea" is displayed. A white cursor block is positioned under the letter 'u' in the word "una".

Además en esa imagen se ve un *prompt* con el contenido `">>> "`. Desde esa situación si el usuario apretara, por ejemplo, la tecla equis esta letra debería insertarse antes de la u. Si apretara la barra de retroceso se debería borrar el espacio, y retroceder todo lo que sigue una posición. La tecla suprimir debería borrar la u que está debajo del cursor y hacer retroceder lo que viene a continuación. Con la tecla inicio el cursor debería ir debajo de la E y la tecla fin debería llevarlo a continuación de la última a.

Este es el comportamiento de cualquier terminal con edición de línea que hayamos usado (la del sistema operativo, la de Python, etc.), abrir una terminal y jugar para familiarizarse con el comportamiento.

Si quisiéramos imprimir una línea de terminal deberíamos primero limpiar la línea completa, luego imprimir el prompt, imprimir el contenido de la línea que se está editando y luego retroceder el cursor para que quede en la posición requerida.

Si quisiéramos editar una línea de terminal, deberíamos tener un arreglo donde guardaremos cada uno de sus caracteres, y además contadores para saber cuántas líneas tenemos y cuántos caracteres mide la línea. Las teclas que ingrese el usuario van a modificar el contenido de la cadena y/o el valor de los contadores.

Eco y buffer

Como ya dijimos, con la entrada estándar de C no podríamos implementar una terminal porque, primero, cada tecla que el usuario ingresara se verá por la pantalla, y hay teclas que queremos capturar para hacer cosas en función de eso y, segundo, porque no queremos esperar a la llegada del `'\n'` para decidir qué hacer.

Tomemos el siguiente código:

```

#include <stdio.h>
#include <termios.h>

int main(void) {
    struct termios config_original;

    if(tcgetattr(0, &config_original) != 0) {
        // No pudo obtenerse la configuración actual de la terminal
        return 1;
    }

    struct termios config_nueva = config_original;
    // Configuramos las opciones de lectura sin buffer y sin eco
    config_nueva.c_lflag &= ~ICANON & ~ECHO;
    if(tcsetattr(0, TCSAFLUSH, &config_nueva) != 0) {
        // No pudimos reconfigurar la terminal
        return 1;
    }

    // Leo un caracter:
    char c = getchar();
    // Si no lo imprimo no lo veo:
    putchar(c);
    // Como no hay buffer no hay que esperar un '\n', esto va a terminar ya.

    // Antes de terminar, recuperamos la configuración original. Si no
    // hacemos esto la terminal va a quedar rota (y sólo la vamos a poder
    // recuperar con el comando reset).
    tcsetattr(0, TCSAFLUSH, &config_original);

    return 0;
}

```

Este código lo que hace es... lee los comentarios, lo dice ahí.

La idea es trabajar configurando la terminal de este modo.

Historial

Dijimos que una terminal tiene una línea para editar con acciones para atrás, adelante, retroceso, suprimir, inicio y fin; pero además una terminal tiene un historial de comandos anteriores. El historial se navega con las teclas arriba y abajo.

Una terminal empieza con una línea en blanco y un historial (que en principio puede estar vacío). Cada vez que una línea se acepta (por ejemplo, al apretar enter) la misma pasa al historial y se crea una nueva línea vacía para seguir ingresando comandos.

Ahora bien, si se apretara la tecla arriba entonces se pasará a editar la última línea, y si se siguiera apretando arriba una más y así. Al aceptar una línea nueva, la que se estaba editando debe ser agregada al historial.

Lectura de teclas especiales

Las teclas especiales que se mencionaron se leen de forma directa o indirecta después de reconfigurar la terminal:

- EOF: Al reconfigurar la terminal el EOF se identifica con el valor `4`,
- Enter: `'\n'`,
- Tecla de retroceso: `127`,
- Izquierda: `91` seguido de `'D'`,
- Derecha: `91` + `'C'`,
- Inicio: `91` + `'H'`,
- Fin: `91` + `'F'`,
- Arriba: `91` + `'A'`,
- Abajo: `91` + `'B'`,
- Suprimir: `91` + `'3'` + una letra más, no importa el valor, si se lee la secuencia 91-`'3'` descartar el siguiente valor.
- A su vez, por fuera de los caracteres ya mencionados queremos descartar todos los valores donde `isprint()` sea `false`.

Trabajo

Colores

Teniendo definido:

```
typedef enum {COLOR_RESET, COLOR_ROJO, COLOR_VERDE, COLOR_AMARILLO, COLOR_AZUL,
              COLOR_MAGENTA, COLOR_CIAN, COLOR_BLANCO} color_t;
```

implementar una función `char *color_escape(color_t c);` que dado un color `c` devuelva la secuencia de escape para pintar la terminal de ese color.

Se deben utilizar tablas de búsqueda para implementar esta función.

Terminal

Teniendo definida la siguiente estructura:

```
#define MAX_LINEA 100

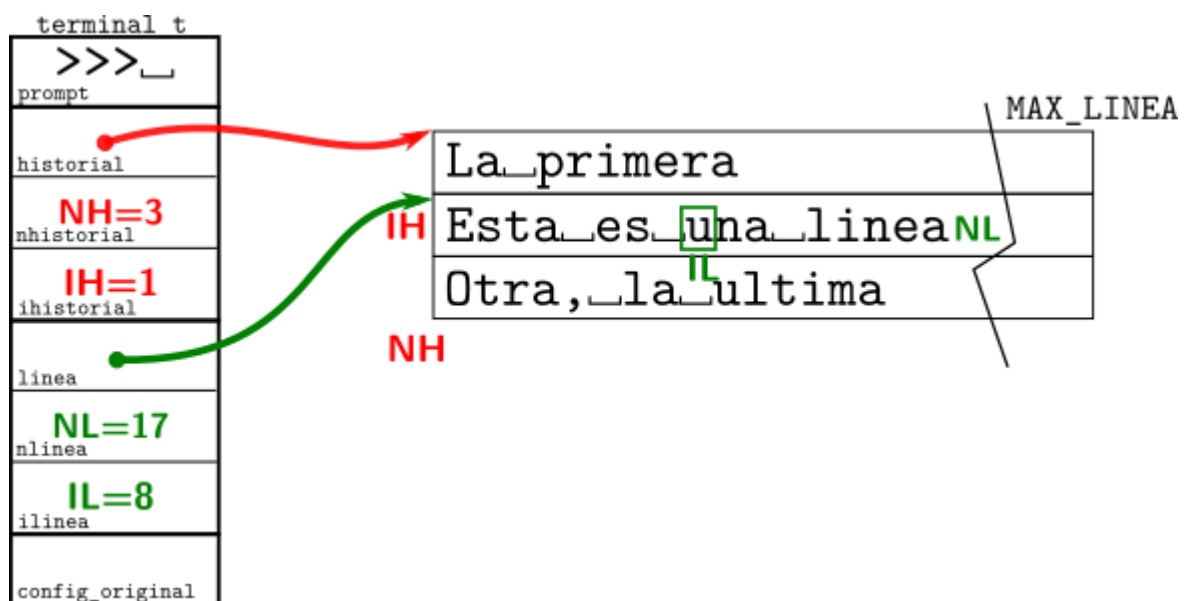
typedef struct {
    // El prompt:
    char prompt[MAX_LINEA];

    // El historial, la cantidad de cadenas y el índice de la línea que se está
    editando:
    char (*historial)[MAX_LINEA];
    int nhistorial, ihistorial;

    // La línea que se está editando, su longitud y la posición del cursor:
    char *linea;
    int nlinea, ilinea;

    // La copia de la configuración de terminal original:
    struct termios config_original;
} terminal_t;
```

Donde, por ejemplo la misma se vería así en memoria:



Siendo una terminal que está editando la línea "Esta es una línea", con el cursor idéntico al del ejemplo anterior, pero además tiene en el historial una línea anterior, la primera, y una nueva línea, la última, que el usuario editó pero luego prefirió seguir editando la anteúltima.

Creación y destrucción

Implementar una función `terminal_t *terminal_crear(const char *prompt);` que cree una estructura de tipo terminal y la devuelva.

La función debe:

- Inicializar el `prompt` con el `prompt` pasado como parámetro.
- Crear un historial de tamaño 1, con una cadena vacía.
- `linea` apuntará a esta cadena del historial.
- Configurar la terminal en modo no eco y sin buffer, guardándose la configuración.

Implementar una función `void terminal_destruir(terminal_t *t);` que libere la memoria del historial, de la estructura y que además restaure la configuración de la terminal.

Lectura

Implementar una función `int terminal_leer(terminal_t *t);` que lea (en principio) un carácter de `stdin` y lo procese. El procesamiento será el que ya se describió en la sección de introducción. Si bien se pide leer "un carácter" en el caso de que se reciba un carácter compuesto el mismo debe ser leído completo con la cantidad de lecturas que correspondan (2 para la mayor parte, 3 para la tecla suprimir).

La función debe devolver `EOF` en caso de haber leído la señal de EOF (ver la lista de teclas especiales), `'\n'` en caso de haber leído el `'\n'`, `0` en el caso de haber leído un carácter no validado por `isprint()` y el carácter leído en cualquier otro caso.

Repasamos el comportamiento a tener para los diferentes caracteres:

- EOF: Devuelve `EOF`,
- Izquierda: Decrementa el cursor (`ilinea`),
- Derecha: Incrementa el cursor,
- Inicio: Cursor a cero,
- Fin: Cursor a longitud de línea (`nlinea`),
- Suprimir: Saca de `linea` el carácter que está en la posición del cursor. Desplaza toda la cadena que continúa uno hacia atrás. Decrementa la longitud de la línea.

- Retroceder: Saca de la línea el caracter que está atrás del cursor. Desplaza la línea, decrementa el cursor y la longitud.
- Arriba: Escribe el `'\0'` en la línea actual. Decrementa el índice de línea en edición (`ihistorial`) y reemplaza `línea` por un puntero a la nueva línea en edición. Actualiza el cursor y la longitud de la línea para pararse al final.
- Abajo: Ídem a arriba pero incrementando el índice.
- Enter: Escribir el `'\0'` en la línea actual. En el caso de que la línea que se estaba editando no fuera la última del historial (`ihistorial != nhistorial - 1`) copiar la línea a la última posición del historial. Luego agregar una nueva línea en blanco al historial y ajustar todo para que esta sea la nueva línea a editar.
- Cualquier otra letra imprimible: La inserta en la línea en la posición actual del cursor desplazando todo lo que viniera después a la derecha. Incrementa el cursor y la longitud.

Prestarle atención a que hay un montón de casos de borde con los índices, por ejemplo, apretar retroceso estando el cursor en `0`, etc. Estos casos deben ser tenidos en cuenta.

La acción de apretar enter realiza una manipulación de memoria. Devolver `EOF` en caso de fallas de memoria.

Impresión de la terminal

Implementar una función `void terminal_imprimir(const terminal_t *t);` que limpie la línea, imprima el prompt, la línea actual y posicione el cursor donde corresponda.

Devolución de una línea del historial

Implementar una función `char *terminal_historial(const terminal_t *t, int i);` que reciba un índice y devuelva la línea del historial que corresponda (o `NULL` en caso de no existir). El índice `0` tiene que devolver la última línea, `-1` la anterior, `-2` la anteúltima y así.

Por ejemplo, después de leído un enter `terminal_historial(t, -1);` debe devolver la última línea ingresada por el usuario.

Programa

Se provee el código fuente de la función `main()` que muestra el uso de esta herramienta:

```
int main(void) {
    terminal_t *t = terminal_crear("$ ");
    if(t == NULL)
        return 1;

    terminal_imprimir(t);

    int c;
    while((c = terminal_leer(t)) != EOF) {
        if(! (rand() % 50))
            printf("\033[2K\033[1G%s<<< Hola, soy texto interrumpiendo!\n%s",
color_escape(COLOR_ROJO), color_escape(COLOR_RESET));

        if(c == '\n') {
            printf("\033[2K\033[1G%s>>> %s\n%s", color_escape(COLOR_VERDE),
terminal_historial(t, -1), color_escape(COLOR_RESET));
        }
        terminal_imprimir(t);
    }

    terminal_destruir(t);
    return 0;
}
```

Entrega

Deberá entregarse el código fuente del programa desarrollado.

El programa debe:

1. Compilar correctamente con los flags:

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

2. pasar Valgrind correctamente,

La entrega se realiza a través del [sistema de entregas](#).

El ejercicio es de entrega individual.