Algoritmos y Estructuras de Datos II

Laboratorio - 13/03/2025

Laboratorio 0: Repaso del lenguaje C

- Revisión 2024: Marco Rocchietti

- Revisión 2025: Franco Luque

Código

lab00-kickstart.tar.gz

Objetivos

1. Repaso de programación básica en lenguaje C
2. Repaso de compilación de programas
3. Uso de editor de texto para trabajar
4. Repaso de arreglos y estructuras en C.
5. Repaso de uso de las librerías **`assert.h`**, **`stdio.h`**, **`stdbool.h`**
6. Lectura y comprensión de código.

Requerimientos

1. Compilar con los flags de la materia:
   ```
   $ gcc -Wall -Werror -Wextra -pedantic -std=c99 …
   ```
2. Seguir las guías de estilo
3. Prohibido usar break, continue y goto!!
4. Prohibido usar return a la mitad de una función.

Recursos

Recursos generales:

- Videos del Laboratorio en el aula virtual
- Documentación en el aula virtual
- Estilo de codificación:
  - Guía de estilo para la programación en C
  - Consejos de Estilo de Programación en C

Recursos específicos:

- Lab 0: Resolviendo is_upperbound
- Condicional en C: if

- Ciclos en C: while y for
- Función scanf
- Biblioteca stdbool.h
- Páginas del manual de linux:
    - Comando “man stdbool”

Ejercicio 1

En el archivo **bounds.c** implementar la función:

```
struct bound_data check_bound(int value, int arr[], unsigned int length);
```

que determina si por un lado el valor `value` es mayor o igual a todos los elementos del arreglo `arr` (que tiene `length` elementos), por otro lado si es menor o igual a todos los elementos del arreglo, también indica si `value` se encuentra en `arr` y en caso de encontrarse indica en qué posición se lo encontró. Toda esta información es devuelta en una estructura **struct** `bound_data` que tiene la siguiente definición:

```
struct bound_data {
    bool is_upperbound;
    bool is_lowerbound;
    bool exists;
    unsigned int where;
};
```

Los cuatro campos de la estructura son los siguientes: valor de verdad que indica si el valor es mayor o igual a todos los elementos del arreglo (`is_upperbound`), valor de verdad que indica si el valor es menor o igual a todos los elementos (`is_lowerbound`), valor de verdad que indica si el elemento existe en el arreglo (`exists`) y posición donde se encontró el elemento (`where`). Por ejemplo

```
int a[] = {0, -1, 9, 4};
result = check_bound(9, a, 4);
printf("%d", result.is_upperbound) // Imprime 1
printf("%d", result.is_lowerbound) // Imprime 0
printf("%d", result.exists)        // Imprime 1
printf("%u", result.where)         // Imprime 2
```

La función debe implementarse usando un único ciclo (`for` o `while`).

En la función **main()** se le debe solicitar al usuario que ingrese uno por uno los elementos del arreglo, para ello utilizar la función **scanf()** de la librería **stdio.h**. Una vez que se obtienen los elementos, se le debe pedir al usuario un valor para verificar con **check_bound()**. La salida por pantalla debe indicar si el valor que ingresó el usuario es *cota superior*, *cota inferior*, *mínimo* o *máximo*. Solo en caso de ser mínimo o máximo (esto significa que el valor se encuentra en el arreglo) mostrar por pantalla la posición donde se encontró el elemento

Ejercicio 2b

En el archivo **`tictactoe.c`** se encuentra una implementación incompleta del clásico juego tres en línea (conocido como *tatetí* o *tictactoe*). El tablero *3x3* se representa con una matriz en C, declarada de la siguiente manera.

```
char board[3][3] = {
        { '-', '-', '-' },
        { '-', '-', '-' },
        { '-', '-', '-' }
};
```

Inicialmente todas las “celdas” del tablero se encuentran vacías, lo que se representa con el carácter **`'-'`**. Los caracteres **`'X'`** y **`'O'`** se utilizan para representar la cruz y el círculo, respectivamente. El juego permite al usuario llenar una celda del casillero con **`'X'`** y **`'O'`** de acuerdo a su turno, teniendo la posibilidad de elegir con un número entero en qué celda desea marcar el tablero. Las nueve celdas del tablero se encuentran numeradas de la siguiente forma:

```
.............................................
| 0: -          | 1: -          | 2: -          |
.............................................
| 3: -          | 4: -          | 5: -          |
.............................................
| 6: -          | 7: -          | 8: -          |
.............................................
```

El juego está incompleto puesto que no detecta cuándo hubo un ganador, o si hubo empate. Para ello se deben implementar correctamente las funciones:

```
bool has_free_cell(char board[3][3])
```

que devuelve verdadero si hay una celda libre (marcada con **`'-'`**) en el tablero **`board`**, y devuelve **`false`** en caso contrario;

```
char get_winner(char board[3][3])
```

que devuelve el jugador ganador (**`'X'`** o **`'O'`**) si lo hubo, o **`'-'`** si todavía no hay ganador. Para ello se debe recorrer la matriz para verificar si alguna columna, fila o diagonal tiene 3 veces consecutivas el mismo carácter.|

Se pide leer el código y comprender cómo se logra el funcionamiento del juego.

Ejercicio 3

a) En un archivo nuevo **`tictactoe_gen.c`** programar una implementación modificada del ejercicio anterior para hacer un tictactoe que utilice un tablero 4x4 donde el criterio para ganar es que haya 4 marcas iguales en línea en vez de 3.

b) Modificar la implementación del apartado anterior (en el mismo archivo) para implementar un juego de tictactoe en un tablero 5x5. Si hicieron correctamente el apartado *a)* debería salir cambiando un solo símbolo del código. Si requieren más cambios, rehacer el apartado *a)*