Examen final 2022-02-14

95.11 - Algoritmos y Programación I - Curso Essaya

Objetivo

Se dispone de los siguientes archivos:

- Archivos provistos con código:
  - `Makefile`
- Archivos a completar con código:
  - `ej1.c`, `ej2.c`, `ej3.c`, `ej4.c`, `ej5.c`: Implementación + pruebas de cada ejercicio

Al compilar con `make`, se genera un archivo ejecutable para cada ejercicio: `ej1`, `ej2`, etc. Cada uno de ellos, al ejecutarlo, corre las pruebas para verificar el correcto funcionamiento de la implementación.

El examen se aprueba con al menos 3 ejercicios correctamente resueltos. Un ejercicio se considera correctamente resuelto si:

- El programa `ej<n>` compila sin advertencias ni errores
- La implementación cumple con lo pedido en el enunciado

En algunos ejercicios se incluye un ejemplo de uno o dos casos de prueba y queda a cargo del alumno agregar más casos de prueba, para los que se provee sugerencias. En otros ejercicios se provee únicamente sugerencias. La implementación de las pruebas adicionales es **opcional**, pero se recomienda hacerlo ya que permite asegurar que la resolución del ejercicio es correcta.

Makefile

Para compilar el ejercicio `<n>`: `make ej<n>`. Por ejemplo, para compilar y ejecutar el ejercicio 1:

```
$ make ej1
gcc -Wall -pedantic -std=c99 ej1.c -o ej1
$ ./ej1
ej1.c: OK
```

Salida del programa

Al ejecutar `./ej<n>` se imprime el resultado de las pruebas. Si todas las pruebas pasan correctamente, se imprime `OK`. En caso contrario, cuando una de las verificaciones falla, se imprime un mensaje de error y el programa termina su ejecución. Por ejemplo:

```
$ ./ej1
ej1: ej1.c:45: main: Assertion `p != NULL' failed.
sh: “./ej1” terminated by signal SIGABRT (Abort)
```

Recordar: Correr cada uno de los ejercicios con `valgrind --leak-check=full` para mayor seguridad de que la implementación es correcta.

Pruebas

Se recomienda usar la función `assert` de la biblioteca estándar para verificar condiciones en las pruebas. Ejemplo de uso:

```c
#include <stdio.h>
#include <assert.h>

// funcion a probar
int sumar(int a, int b) {
    return a + b;
}

// pruebas
int main(void) {
    assert(sumar(0, 0) == 0);
    assert(sumar(2, 3) == 5);
    assert(sumar(2, -2) == 0);

    printf("%s: OK\n", __FILE__);
    return 0;
}
```

Nota: A veces para depurar un error en las pruebas es útil imprimir valores; se permite el uso de `printf()` para ello.

Nota: A veces para implementar las pruebas es útil utilizar números aleatorios. Se permite el uso de `rand()` para ello. En ese caso, se recomienda ejecutar `srand(0);` al inicio del programa para asegurar que la secuencia de números aleatorios sea siempre la misma, y así facilitar la depuración.

Ejrcicios

Ejercicio 1 Escribir la función `void patron_tablero(int n, bool m[n][n], int k)`, que recibe una matriz de `n x n` booleanos y un número `k` (con `k < n`), y escribe en la matriz un patrón de tablero de ajedrez donde cada casillero es de tamaño `k x k`.

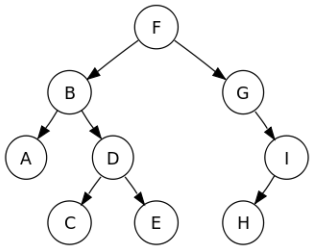
Ejemplo: luego de invocar `patron_tablero(10, m, 3)`, la matriz `m` debe quedar de la siguiente manera (0 y 1 representando `false` y `true` respectivamente):

```
0001110001
0001110001
0001110001
1110001110
1110001110
1110001110
0001110001
0001110001
0001110001
1110001110
```

Ejercicio 2 Se tiene una señal de audio almacenada como una secuencia de números `float` en un archivo binario. Implementar un programa que reciba por argumento de línea de comandos un número entero no negativo `V` y dos nombres de archivo `E` y `S`. El programa debe leer de `E` la señal de audio, aplicar a cada muestra un cambio de volumen (multiplicar por `V`) y guardar en `S` el resultado.

El valor de `V` se debe interpretar como un porcentaje, de forma tal de que con `V = 100` la señal producida queda igual a la original.

Ejercicio 3 Un *árbol binario* es una estructura enlazada en la que cada nodo contiene referencias a otros dos nodos, llamados *hijo izquierdo* y *derecho* (pudiendo cualquiera de ellos ser una referencia nula).



Se denomina *raíz* al primer nodo del árbol, y *hojas* a aquellos nodos del árbol que no tienen hijo izquierdo ni derecho.

Ejemplo: para el árbol mostrado arriba, la raíz es el nodo `F`, y las hojas son los nodos `A`, `C`, `E` y `H`.

Dada la estructura `nodo_t` que representa un nodo del árbol, se pide implementar la función `int cantidad_hojas(nodo_t *raiz)`, que recibe el nodo raíz y devuelve la cantidad de hojas que contiene el árbol.

Sugerencia: pensar la función en forma recursiva.

Ejercicio 4 Sea la estructura `alumno_t` que representa a un alumno de FIUBA:

```
typedef struct {
    char *nombre;
    int padron;
} alumno_t;
```

a. Implementar las funciones `alumno_crear` y `alumno_destruir`.

b. Implementar la función `alumno_t *buscar_alumno(char *nombre, alumno_t *alumnos[], size_t n)` que busca en el listado de alumnos recibido el alumno con el nombre indicado y devuelve un puntero al mismo, o NULL en caso de no encontrarlo. La lista de alumnos está ordenada por nombre (orden lexicográfico según el valor de la tabla ASCII), y el algoritmo de búsqueda utilizado debe ser **mejor que lineal**.

Ejercicio 5 En un software de procesamiento de texto, el texto tiene un *estilo* y un *tamaño*. El estilo se determina por una combinación de las siguientes *banderas*:

- Negrita
- Cursiva
- Subrayado
- Tachado

El tamaño, a su vez, es un número entero no negativo, entre 0 y 4095.

Todas estas propiedades deben ser almacenadas en un registro de 16 bits:

```
typedef uint16_t prop_t;
```

Diseñar la composición de dicho registro (es decir, qué bits representan qué cosa), e implementar las funciones:

```
* ``void prop_set_negrita(prop_t *reg, bool negrita)``
* ``bool prop_get_negrita(prop_t *reg)``
* ``void prop_set_cursiva(prop_t *reg, bool cursiva)``
* ``bool prop_get_cursiva(prop_t *reg)``
* ``void prop_set_subrayado(prop_t *reg, bool subrayado)``
* ``bool prop_get_subrayado(prop_t *reg)``
* ``void prop_set_tachado(prop_t *reg, bool tachado)``
* ``bool prop_get_tachado(prop_t *reg)``
* ``void prop_set_tamanio(prop_t *reg, unsigned int tamanio)``
* ``unsigned int prop_get_tamanio(prop_t *reg)``
```