Nombre y Apellido: ..... N° Legajo: .....

Primer Parcial de Programación Imperativa

19/04/2024

- **Condición mínima de aprobación: Sumar 5 (cinco) puntos.**
- **Los ejercicios que no se ajusten estrictamente al enunciado, no serán aceptados.**
- **No usar variables globales ni static.**
- **No es necesario escribir los #include**
- **Escribir en esta hoja Nombre, Apellido y Legajo**

Ejercicio 1 (3 puntos)

Escribir la función **`firstN`** que recibe un vector de **unsigned chars**, su **dimensión** y un valor **entero no negativo n**. La función debe dejar en el vector **las primeras n apariciones de cada valor** y devolver en dos parámetros de salida:

- la dimensión del nuevo vector
- cuántos elementos eliminó

Ejemplo de uso:

```
int main(void) {
   unsigned char v[] = {1, 2, 1, 3, 1, 4, 5, 2};
   int dim, del;
   firstN(v, 8, 3, &dim, &del); // dim es 8, del es 0, v no cambia
   firstN(v, 8, 8, &dim, &del); // dim es 8, del es 0, v no cambia
   firstN(v, 8, 4, &dim, &del); // dim es 8, del es 0, v no cambia
   firstN(v, 8, 1, &dim, &del); // dim=5, del=3, v = {1, 2, 3, 4, 5};

   unsigned char w[] = {1, 2, 1, 3, 1, 4, 5, 2};
   firstN(w, 8, 0, &dim, &del); // dim=0, del=8, w = {}

   unsigned char t[] = {1, 2, 1, 3, 1, 4, 5, 2};
   firstN(t, 8, 2, &dim, &del); // dim=7, del=1, t = {1, 2, 1, 3, 4, 5, 2}

   return 0;
}
```

Ejercicio 2 (3,50 puntos)

Escribir la función **`nQueens`** que, dada una matriz de chars de `NxN`, verifique si el mismo **contiene N reinas de ajedrez** de forma tal que **ninguna reina amenace a otra**. Una reina amenaza a otra **si está en la misma fila, columna o en alguna de sus dos diagonales**. Una posición libre se indica con el valor '0' y una celda con una reina se indica con el valor '1'. Se asume que en el tablero sólo hay caracteres ceros y unos, no es necesario validarlo.

Programa de prueba:

```
int main(void) {
    char board[][8] = {
            {'0', '0', '0', '0', '1', '0', '0', '0'},
            {'0', '0', '1', '0', '0', '0', '0', '0'},
            {'0', '0', '0', '0', '1', '0', '0', '0'},
            {'0', '0', '0', '0', '0', '0', '0', '0'},
            {'0', '1', '0', '0', '0', '0', '0', '0'},
            {'0', '0', '0', '0', '0', '0', '1', '0'},
            {'1', '0', '0', '0', '1', '0', '0', '0'},
            {'0', '0', '0', '0', '0', '0', '0', '1'},
    };
    // por ejemplo, hay tres reinas en la columna 4 y dos en la fila 6
    assert(nQueens(8, board) == 0);

    char board2[][6] = {
            {'0', '0', '0', '1', '0', '0'},
            {'1', '0', '0', '0', '0', '0'},
            {'0', '0', '0', '0', '1', '0'},
            {'0', '1', '0', '0', '0', '0'},
            {'0', '0', '0', '0', '0', '1'},
            {'0', '0', '1', '0', '0', '0'}
    };
    // Hay 6 reinas y no se atacan, retorna true
    assert(nQueens(6, board2));

    char board3[][6] = {
            {'0', '0', '1', '0', '0', '0'},
            {'1', '0', '0', '0', '0', '0'},
            {'0', '1', '0', '0', '0', '0'},
            {'0', '0', '0', '0', '1', '0'},
            {'0', '0', '0', '0', '0', '0'},
            {'0', '0', '0', '0', '0', '1'}
    };
    // Las reinas en (1,0) y (2,1) se amenazan en forma diagonal
    assert(nQueens(6, board3) == 0);

    char board4[][6] = {
            {'0', '0', '0', '1', '0', '0'},
            {'0', '0', '0', '0', '0', '0'},
            {'0', '0', '0', '0', '1', '0'},
            {'0', '1', '0', '0', '0', '0'},
            {'0', '0', '0', '0', '0', '1'},
            {'0', '0', '1', '0', '0', '0'}
    };
    // No se amenazan pero hay menos de 6 reinas
    assert(nQueens(6, board4) == 0);

    // No hay solución para tableros de dimensión 2 o 3
    assert(nQueens(3, board2) == 0);
    assert(nQueens(2, board2) == 0);

    // En un tablero de 1x1 debe haber una reina
    char board1[][1] = {{'1'}};
    assert(nQueens(1, board1));

    puts("OK!");
    return 0;
}
```

Ejercicio 3 (3,50 puntos)

Una **escalera de palabras** (Word Ladder) consiste en un conjunto de palabras donde **cada una difiere de la anterior únicamente en una letra** (sin distinguir mayúsculas de minúsculas) en una posición.

A partir de una primera palabra `TEST` se puede conseguir la palabra `BEST` cambiando la primer letra, y luego se puede conseguir `BEET` cambiando la tercer letra de la anterior. De esta forma `TEST -> BEST -> BEET` es una escalera de palabras.

No se forma una escalera de palabras si:

- Cambian dos o más letras de una palabra a otra. Por ejemplo `TEST -> EAST` cambia `T->E` y `E->A`
- La posición de la letra que cambió se repite entre dos palabras **seguidas**. Por ejemplo `TEST -> BEST -> REST` cambia dos veces seguidas la primera letra `T->B` y `B->R`.

Implementar una función **`isWordLadder`** que recibe:

- la cantidad de palabras del conjunto
- la longitud de cada una de las palabras (Se asume que todas son de la misma longitud, no es necesario validarlo)
- una matriz de chars donde cada fila de la matriz corresponde a una palabra

y retorna 1 si la matriz corresponde a una escalera de palabras y 0 sino.

Ejemplo de uso:

```
int main(void) {
    char wordMatrix[][COLS] = {
            {'T', 'e', 'S', 't'},
            {'b', 'e', 'S', 'T'},
            {'B', 'E', 's', 'o'},
            {'B', 'E', 'S', 'A'}};
    // Considerando todas las filas no es escalera
    // pues los últimos dos cambios son en la última letra
    assert(!isWordLadder(4, 4, wordMatrix));
    // Considerando una sola fila es escalera
    assert(isWordLadder(1, 4, wordMatrix) == 1);
    // Considerando las tres primeras filas es escalera
    assert(isWordLadder(3, 4, wordMatrix) == 1);

    char wordMatrix2[][3] = {
            {'T', 'e', 'S'},
            {'t', 'b', 'e'},
            {'S', 'b', 'E'},
            {'B', 'b', 'e'}};
    // TeS -> Tbe cambian dos letras
    assert(!isWordLadder(4, 3, wordMatrix2));

    char wordMatrix3[][3] = {
            {'a', 'm', 'o'},
            {'a', 'r', 'o'},
            {'o', 's', 'a'}};
    // aro -> osa cambian tres letras
    assert(!isWordLadder(3, 3, wordMatrix3));
    puts("OK!");
    return 0;
}
```