TP3-2024

El objetivo de esta práctica es que refuercen el manejo de arreglos y trabajen con cadenas de caracteres.

Forma de entrega

- Se indicará un par de ejercicios que deberán ser subidos al campus para su verficación.
- Cada punto debe ser entregado en un archivo independiente.
- El nombre de cada archivo debe ser `ejercicio` seguido del número de ejercicios más `.c`, de esta manera, el primer ejercicio será entonces `ejercicio1.c`.
- Recuerden tener en cuenta las **Cuestiones de Estilo**
- En ningún caso se aceptará el uso de variables globales. Toda la información necesaria para el funcionamiento de las funciones a desarrollar tienen que ser pasado como argumentos de las mismas.
- Mantengan separado lo que es entrada, del algoritmo y la salida.
- No olviden documentar las funciones implementadas indicando el propósito de los argumentos y que es lo que retorna.

Ejercicios

Punto 1. Cadenas seguras

Manipular un arreglo de manera segura es poder limitar la cantidad de posiciones que serán recorridas en el caso de que la cadena de caracteres no cuente con su carácter de terminación (`\0`). Es decir, que:

- El tamaño de los arreglos debe ser determinado en tiempo de compilación con un tamaño máximo.
- El tamaño máximo debe incorporarse en las funciones como un parámetro explícito.
- El tamaño máximo debe limitar el recorrido en las cadenas.

Todas las funciones deben retornar códigos de error como números negativos para las situaciones que se puedan encontrar, como que la cadena de destino es más chica de lo que se necesita. Por ejemplo:

```
#define CADENA_SIN_TERMINADOR -1
```

Siendo que el ejercicio consiste en crear funciones de cadenas, **no esta permitido el uso de funciones de la librería estandar como `string.h`**.

Implementar las funciones y mostrar uno o dos casos de prueba no triviales.

1.1 Largo de cadenas

Implementar una función que cuente el largo de una cadena de texto de manera segura, La función debe retornar el largo de la cadena o el código de error correspondiente.

Este prototipo es un ejemplo para el resto de las funciones a implementar.

```
/**
 * La funcion cuenta los caracteres de la cadena sin exceder la capacidad
 *  @param cadena es la secuencia de caracteres de la cual se obtendrá el
            La secuencia es valida y posee un tamaño físico que permite
 *  @param capacidad es cuantos caracteres pueden ser alojados en la caden
 *                  Este valor es mayor que cero y menor o igual al largo
 *  @returns un numero entero en donde los valores positivos incluyendo el
        y los valores negativos, las siguientes situaciones de error:
           CADENA_SIN_TERMINADOR cuando la cantidad de carateres sea igua
              indicada por el argumento
 */
int largo_seguro(size_t capacidad, char cadena[]);
```

1.2 Copia de cadenas

Implementar una función que copie una cadena en otra y deje solo un `\0` en la cadena destino

La función debe retornar el largo de la cadena destino o el código de error correspondiente.

1.3 Concatenación de cadenas

Implmentar una función que reciba 2 cadenas de caracteres y las concatene en la primera.

concat("hola","mundo")->"holamundo"

1.4 Inserción de cadenas

Implementar una función que inserte una cadena en otra luego de la posición indicada de manera segura.

Que hacer cuando la cadena destino no tiene la capacidad necesaria para alojar la cadena a insertar es su decisión, la cual debe quedar registrada en el comentario de la función.

Ejemplo

Insertar "HOLA" en "Mundo" en la posición 2 daría como resultado "MunHOLAdo"

La función debe retornar el largo de la cadena destino o el código de error correspondiente.

1.5 Limpieza

Implementar una función que limpie la cadena de todos los caracteres que no sean los alfanuméricos (`AZaz09`) menos el `\0` en una cadena diferente a la de entrada.

La función debe retornar el largo de la cadena destino o el código de error correspondiente.

1.6 Comparación

Implementar una función que indique el ordenamiento alfabético de dos cadenas (solo minúsculas)

- -1 la primera cadena va antes de que las segunda
- 0 ambas cadenas son iguales
- 1 la primera cadena va después que la segunda.

1.7 A minúsculas y a mayúsculas

Implementar una función que modifique la cadena dejando todos sus caracteres en minúsculas.

1.8 De vocales a números

Implementar una función que transforme una cadena de caracteres reemplazando las vocales por números según: A/a->4, E/e->3, I/i->1, O/o -> 0, U/u->6.

1.9 Palíndromo seguro

Implementar una función que indique con TRUE si la cadena es un palíndromo, con FALSE si no lo es o indicar un error si no posee `\0`.

Una palabra es palíndromo cuando se puede leer de la misma manera en ambas direcciones.

Ignoren mayúsculas y minúsculas.

Neuquen -> es palíndromo

Opcionalmente, pueden agregar un argumento para las opciones de la función, como ignorar Mayúsculas/minúsculas o símbolos.

Punto 2. Arreglos

2.1 Búsquedas:

Implementar funciones que, dado un arreglo, devuelva:

1. Mínimo
2. Máximo
3. Elemento particular pasado por parámetro.

2.2 Recorridos:

Implementar funciones que, dado un arreglo imprima los valores del arreglo según:

1. De atrás para adelante.
2. Sólo los elementos pares o impares, según un argumentos que será 0 (pares) o 1 (impares).
3. Elemento particular pasado por parámetro.
4. Que tome 2 arreglos y recorra el primero siguiendo los elementos del segundo. Por ejemplo:
    - recorre([1,2,3,4,5,6], [4,0,2]) -> imprime: "5", "1", y "3"
    - recorre([1,2,3,4,5,6], [4,8,2]) -> imprime: "5", "Error: índice fuera de rango"

2.3 Ordenamientos:

1. Implementar un algoritmo que, dado un arreglo y un parámetro ">" (mayor) o "<" (menor) ordene un arreglo de menor a mayor o al revés.