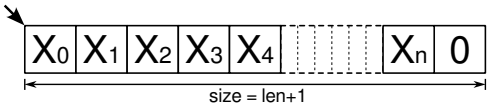
Práctica 1 - Programación en C

Tecnología Digital II

Los siguientes ejercicios están ideados para ser realizados en lenguaje C. Ya sea que se realicen en papel o en computadora, se proveen archivos complementarios para ejecutar y resolver cada uno de los ejercicios. Para compilar los archivos complementarios se puede utilizar el comando `make` que construye los ejecutables de todos los ejercicios.

Ejercicio 1

El lenguaje C no posee definido el tipo básico *string*, como sucede en otros lenguajes de programación. Sin embargo, podemos implementar un *string* como una cadena de caracteres, es decir, un arreglo de caracteres. Para identificar dónde termina la cadena de caracteres, escribimos un 0 (cero) como último carácter de la cadena. Por lo tanto, se dice que un *string* de C es un arreglo de caracteres cuyo último carácter es un 0.



Notar que en ejemplo `size` representa la cantidad de bytes que ocupa la *string*, mientras que `len`, la cantidad de caracteres válidos.

Considerando un *string* de C, construir las siguientes funciones de *strings*:

1. `int len(char* s)`: Toma un *string* y retorna su longitud sin considerar el carácter nulo de terminación.
   Ejemplos:
   `len("Hola")` → `4`
   `len("Chau\n")` → `5`
2. `char* copy(char* s)`: Toma un *string* y genera una copia del mismo.
3. `void replaceChar(char* s, char old, char new)`: Toma un *string* y remplaza todas las apariciones del carácter `old` por `new`.
   Ejemplo: `replaceChar("Hola Carola","l","t")` → `"Hota Carota"`
4. `char* concatenate(char* s1, char* s2)`: Toma dos *strings* y genera un nuevo *string* que contiene la concatenación de ambos (primero `s1` y luego `s2`). La memoria de los *strings* pasados por parámetro debe ser liberada.
   Ejemplo: `concatenate("Guita","rra")` → `"Guitarra"`

Ejercicio 2

1. Implementar una función que tome dos arreglos de enteros (`int`) ordenados de menor a mayor y retorne un nuevo arreglo que combine ambos arreglos de forma ordenada. La función se debe llamar `int* merge(int* A, int sizeA, int* B, int sizeB)`, donde `A` y `B` son punteros a arreglos y `sizeA` y `sizeB` sus respectivos tamaños.
   Ejemplo: `merge([2,3,5,7,9,9], 6, [8,10,12], 3)` → `[2,3,5,7,8,9,9,10,12]`
2. Modificar el ítem anterior para que los números de los arreglos sean de tipo `float`.

Ejercicio 3

Considerar un arreglo de punteros a *string*.

1. Implementar la función `char* longest(char* v[], int size)`, que toma un arreglo de punteros a *string* y retorna el puntero al string más largo. De existir más de un *string* que cumpla la condición, se debe retornar el primero en aparecer en el arreglo.
   Ejemplo: `longest(["o","Hola","Pepe","Eh"], 4)` → `"Hola"`

2. Implementar la función `char* superConcatenate(char* v[], int size)`, que toma un arreglo de punteros a *string* y retorna la concatenación de todos los elementos del arreglo.
   Ejemplo: `superConcatenate(["Eh_","oo oo","+Gol"], 3)` → `"Eh_oo oo+Gol"`

3. Modificar la función anterior para soportar un nuevo parámetro de tipo *string*. Este parámetro se utilizará para concatenarse entre cada par de string del arrelgo. La aridad de esta función será: `char* superConcatenateWithSep(char* v[], int size, char* s)`.
   Ejemplo: `superConcatenateWithSep(["Hola","Pepe","Tito"], 3, "..")` → `"Hola..Pepe..Tito"`

Ejercicio 4

Implementar la función `void pairOfEquals(char v[], int size, char** a, char** b)` que busca dentro de `v` dos bytes iguales y escribe en `a` y `b` los punteros a los bytes encontrados. En el caso de no encontrar ningún par de bytes iguales debe retornar cero en ambos punteros.
Recordar que el tipo `char` es de 1 byte de tamaño y se opera como un número con signo.
Ejemplo: `pairOfEquals([14,23,55,21,55,53], &a, &b)` → `a = &v[2]` y `b = &v[4]`
Ejemplo: `pairOfEquals([14,23,22,21,55,53], &a, &b)` → `a = 0` y `b = 0`

Ejercicio 5

Suponer un puntero a un arreglo de 12 bytes.

1. Interpretar al arreglo como un puntero a `char*` (enteros de un byte) y calcular la sumatoria de los datos almacenados. La aridad de la función es `int sumAsBytes(char* v)`
   Ejemplo: `sumAsBytes( [0x05,0x01,0x00,0x00,0x05,0x01,0x00,0x00,0x05,0x01,0x00,0x00] )` → `0x05+0x01+0x00+0x00+0x05+0x01+0x00+0x00+0x05+0x01+0x00+0x00` → `0x12`

2. Interpretar al arreglo como un puntero a `int*` (enteros de 4 bytes) y calcular la sumatoria de los datos almacenados. La aridad de la función es `int sumAsInts(int* v)`
   Ejemplo: `sumAsInts( [0x05,0x01,0x00,0x00,0x05,0x01,0x00,0x00,0x05,0x01,0x00,0x00] )` → `0x00000105+0x00000105+0x00000105` → `0x30F`

Ejercicio 6

Suponer la siguiente estructura de nodos para una lista:

```
struct node {
    int data;
    struct node *next;
};
```

1. Implementar la función `struct node* addLast(struct node* n, int data)`, que toma un puntero al primer nodo de la lista y agrega en el último lugar de la lista un nuevo nodo que contiene el dato pasado por parámetro. La función debe retornar el puntero al primer nodo de la lista.

2. Implementar la función `struct node* removeFirst(struct node* n)`, que toma un puntero al primer nodo de la lista, borra el primer nodo de la lista y retorna el puntero al nuevo primer elemento de la lista (el que antes era el segundo elemento).

3. Implementar la función `struct node* join(struct node* n1, struct node* n2)`, que toma dos punteros a listas y las concatena: primero `n1` y luego `n2`. La lista resultado utilizará los nodos de las listas pasadas por parámetro. Esta función debe retonar el puntero al primer elemento de la lista resultado.

4. Implementar la función `struct node* removeData(struct node* n, int data)`, que toma un puntero a una lista y un dato, y borra todas las apariciones de `data` en la lista. La función debe retornar el puntero a la lista resultado.

Ejercicio 7

Considerar las siguientes estructuras:

```
struct list {
    struct node *first;
    int size;
};
```

```
struct node {
    struct node *next;
    struct elem *data;
};
```

```
struct elem {
    float x;
    float y;
    int   i;
    char  t;
};
```

La estructura *list* contiene un puntero al primer nodo de una lista de tipo `node`. Cada nodo de la lista contiene un puntero de tipo `elem` con 4 valores numéricos.

1. Implementar la función `float magnitudeAverage(struct list* ls)` que, dada una lista, retorna el promedio de los módulos de los vectores dados por `x` e `y` ($\sqrt{x^2 + y^2}$).

2. Implementar la función `int sorted(struct list* ls)` que, dada una lista, determina si los valores `i` en los nodos están ordenados de forma ascendente.

3. Implementar la función `void numerate(struct list* ls)` que, dada una lista, sobreescribe los valores `t` con el índice que ocupa cada nodo en la lista.

4. Implementar la función `void swap(struct list* ls, int i, int j)` que, dada una lista y dos índices de elementos dento de la lista, intercambia los nodos determinados por los índices dados.

Ejercicio 8

Considerar las siguientes estructuras:

```
struct list {
    struct node *first;
    int size;
};
```

```
struct node {
    struct node *next;
    struct node *prev;
    char *data;
};
```

La estructura `node` contiene dos punteros a nodos: uno al siguiente y otro al anterior. Este tipo de estructura se denomina lista doblemente enlazada.

1. Implementar la función `struct list* addFirst(struct list* ls, char* data)` que agrega un dato como primer elemento de la lista y retorna un puntero a la lista resultado.

2. Implementar la función `struct list* removeFirst(struct list* ls)` que borra el primer elemento de la lista y retorna un puntero a la lista resultado.

3. Implementar la función `struct list* removeNode(struct list* ls, struct node* n)` que, dada una lista y un puntero a un nodo particular de la lista, se ocupa de borrar el nodo pasado por parámetro. Debe retornar un puntero a la lista resultado.

4. Implementar la función `struct list* removeNodei(struct list* ls, int i)` que, dada una lista y un índice dentro del rango de la lista, se ocupa de borrar el i-ésimo nodo de la lista. Si `i<0` o `size-1<i` entonces no hace nada. Debe retornar un puntero a la lista resultado.