Trabajo Práctico 1

Rutas aereas en el mundo terraplanista

Programación en C

Tecnología Digital II

1. Introducción

El objetivo de este TP es implementar un conjunto de funciones sobre distintas estructuras de datos simples y construir además una serie de casos de test para comprobar su correcto funcionamiento. Las funciones a implementar se realizarán sobre dos tipos de datos. Por un lado `strings` de C, es decir, cadenas de caracteres terminadas en *null*, y por otro un nuevo tipo denominado `airTrip`. Este último modelará la lista de paradas de un viaje en avión. Internamente, el tipo `airTrip` utilizará una lista enlazada para guardar datos de cada una de las paradas. Estos datos representarán coordenadas sobre un mundo de cada lugar a visitar.

A fin de simplificar el problema y las operaciones, vamos a considerar al mundo como plano, es decir, las coordenadas de latitud y longitud se moverán en un espacio de coordenadas cartesianas. Esto nos permitirá calcular distancia entre dos puntos usando la raíz de los cuadrados (teorema de Pitágoras) y no tener que calcular el arco del círculo máximo que los une ambos puntos en una esfera.

> El trabajo práctico debe realizarse en grupos de tres personas. Tienen tres semanas para realizar la totalidad de los ejercicios. **La fecha de entrega límite es el 2 de abril hasta las 23:59.** Se solicita no realizar consultas del trabajo práctico por los foros públicos. Limitar las preguntas al foro privado creado para tal fin.

2. Tipo de datos: `airTrip`

Se define a partir de las siguientes estructuras:

```
struct airTrip {
    char* plane;
    float totalLength;
    struct airport* first;
};
```

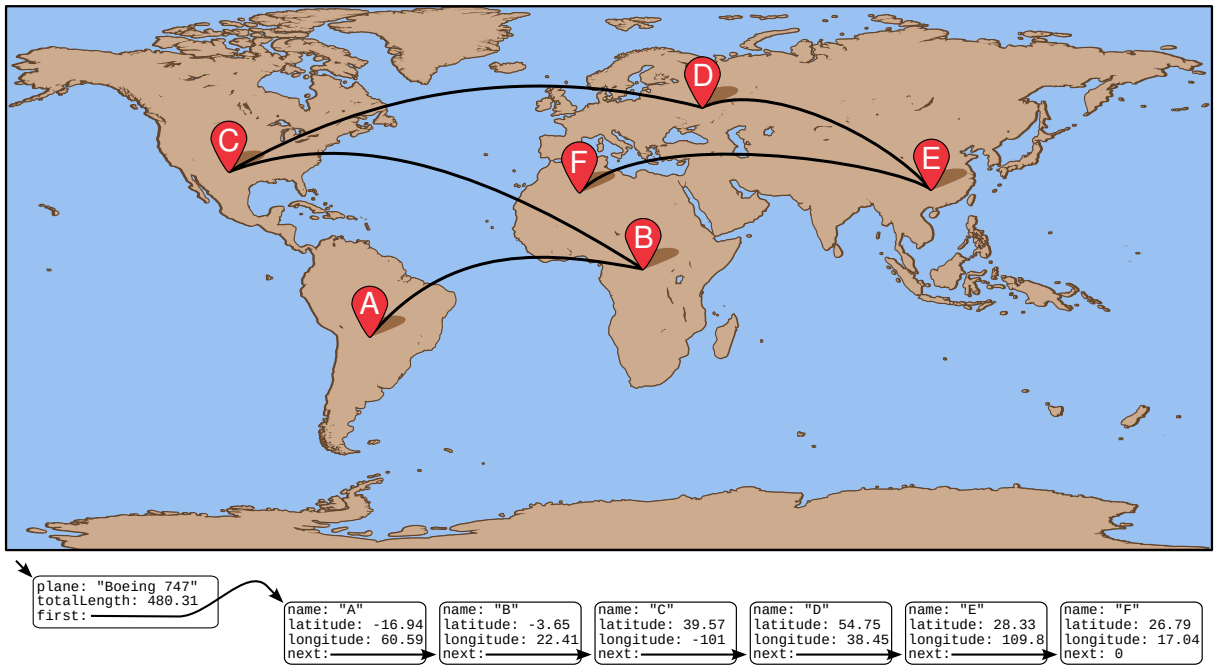
```
struct airport {
    char* name;
    float latitude;
    float longitude;
    struct airport* next;
};
```

La estructura `airTrip`, contiene un puntero a *string* como la identificación de avión, un puntero al primer elemento de la lista de tipo `airport` y la longitud total del recorrido como un dato de tipo `float`.

La lista construida a partir de nodos de tipo `airport` corresponde a las paradas del recorrido del avión, donde cada una se identifica con un nombre como *string*, y un par de coordenadas como valores de tipo `float` que corresponden a los valores de latitud y longitud. Las líneas de latitud toman valores entre -90 y +90 grados, mientras que las coordenadas de longitud están entre -180 y +180 grados.

Si bien la lista puede tener cualquier cantidad de paradas, en el caso de tener cero o una parada, la distancia total del recorrido deberá ser cero. Recién cuando se tenga al menos dos nodos, se podrá calcular la distancia entre cada par y su sumatoria será la longitud total del recorrido.

A continuación se ilustra un ejemplo de la estructura:



3. Enunciado

Ejercicio 1

Implementar las siguientes funciones sobre *strings*.

- `char* strDup(char* src)`
  Duplica un *string*. Debe contar la cantidad de caracteres totales de *src* y solicitar la memoria equivalente. Luego, debe copiar todos los caracteres a esta nueva área de memoria. Además, como valor de retorno se debe retornar el puntero al nuevo *string*.
- `int strCmp(char* s1, char* s2)`
  Compara dos *strings* en orden lexicográfico[1]. Debe retornar:
  - 0 si son iguales
  - 1 si `s1` $<$ `s2`
  - -1 si `s2` $<$ `s1`

  Ejemplos:
  `strCompare("ala","perro")` $\rightarrow$ `1`
  `strCompare("casa","cal")` $\rightarrow$ `-1`
  `strCompare("topo","top")` $\rightarrow$ `-1`
  `strCompare("pato","pato")` $\rightarrow$ `0`

- `char* strCnt(char* src1, char* src2)`

  La función toma dos *strings* `src1` y `src2`, y retorna una nueva *string* que contiene una copia de todos los caracteres de `src1` seguidos de los caracteres de `src2`. Esta función **NO** libera la memoria de las *strings* `src1` y `src2`.

  Ejemplos:
  `strConcatenate("mos","tacho")` $\rightarrow$ `"mostacho"`
  `strConcatenate("elefant","e")` $\rightarrow$ `"elefante"`
  `strConcatenate("tucan","")` $\rightarrow$ `"tucan"`

[1] `https://es.wikipedia.org/wiki/Orden_lexicografico`

Ejercicio 2

Este ejercicio consiste en implementar funciones sobre el tipo de datos `airTrip`. Para simplificar el desarrollo, se provee un conjunto de funciones útiles ya implementadas.

- `struct airTrip* airTripNew(char* plane)`
  Crea una nueva estructura de tipo `airTrip` vacía, sin ninguna parada. Asigna el nombre del avión pasado por parámetro.

- `float flyLength(struct airport* a1, struct airport* a2)`
  Función auxiliar, toma dos punteros a `airport` y calcula la distancia entre ambos.

- `void airTripPrint(struct airTrip* trip)`
  Imprime en pantalla una estructura de `airTrip`, incluyendo la lista de `airport`. La primer linea contiene el nombre del avión junto con el total del recorrido. La segunda linea contiene los nombres de las paradas seguido de sus coordenadas entre paréntesis. Ejemplo:

```
R1234 : 140.01
[T00:(1.00,1.00)][P00:(0.50,0.50)][P01:(1.50,1.50)][T01:(10.00,10.00)][T02:(99.00,99.00)]
```

Se pide entonces implementar las siguientes funciones:

- `void airTripAddLast(struct airTrip* trip, char* name, float longitude, float latitude)`
  Agrega un nuevo `airport` al final de la lista de paradas de un *trip*. Para esto debe recorrer toda la lista hasta encontrar el último lugar donde agregar el nuevo nodo. La nueva estructura de `airport` debe ser completada con los campos pasados por parámetro.
  Ejemplo

```
airTripAddLast(A1:[P1:(1.00,1.00)][P2:(2.00,2.00)], NN, 0.5, 0.5)
            → A1:[P1:(1.00,1.00)][P2:(2.00,2.00)][NN:(0.5,0.5)]
airTripAddLast(A1:[P1:(1.00,1.00)][P2:(2.00,2.00)], NN , 5.00, 5.00)
             → A1:[P1:(1.00,1.00)][P2:(2.00,2.00)][NN:(5.00,5.00)]
airTripAddLast(A1:, NN , 5.00, 5.00)
             → A1:[NN:(5.00,5.00)]
```

- `void airTripAddBest(struct airTrip* trip, char* name, float longitude, float latitude)`
  Agrega un nuevo `airport` en el mejor lugar que se pueda del recorrido. Dejando el primer lugar como fijo, se debe buscar el lugar donde agregar la nueva parada que minimice el recorrido total, es decir, que agregue la menor cantidad de distancia por tener que pasar por esta nueva parada. En los ejemplos a continuación, se puede ver como en el primer caso el mejor lugar para agregar la parada es al final, cambiando el último lugar a visitar. En el segundo caso, lo mejor es que la nueva parada termine entre las únicas dos paradas de la lista. En el último caso, lo mejor debería ser cambiar el primero, pero esto no se permite. Solo se permite buscar el mejor lugar a partir del primero. Por lo tanto, esta nueva parada, se termina agregando en el segundo lugar.
  Ejemplo

```
airTripAddBest(A1:[P1:(1.00,1.00)][P2:(2.00,2.00)], NN, 2.50, 2.50)
             → A1:[P1:(1.00,1.00)][P2:(2.00,2.00)][NN:(2.50,2.50)]
airTripAddBest(A1:[P1:(1.00,1.00)][P2:(2.00,2.00)], NN, 1.50, 1.50)
             → A1:[P1:(1.00,1.00)][NN:(1.50,1.50)][P2:(2.00,2.00)]
airTripAddBest(A1:[P1:(1.00,1.00)][P2:(2.00,2.00)], NN, 0.5, 0.5)
             → A1:[P1:(1.00,1.00)][NN:(0.5,0.5)][P2:(2.00,2.00)]
```

- `void airTripJoin(struct airTrip** tripJoin, struct airTrip* trip1, struct airTrip* trip2)`
  Une dos recorridos borrando los recorridos pasados por parámetro y retornando un nuevo recorrido en el doble puntero. Los recorridos serán unidos directamente colocando todas las paradas del `trip1` y luego todas las paradas del `trip2`. Si el nombre del avión en ambos recorridos es el mismo, entonces se dejará solo este nombre. Si los nombres son diferentes, se deberá construir

una nueva *string* que concatene el nombre del primero con el nombre del segundo separados por un guión. Por ejemplo si el avión era un "Boeing737" para un recorrido y "Boeing747" para el otro recorrido, el nombre resultante será "Boeing737-Boeing747".

Ejemplo

```
airTripJoin(A1:[P1:(1.00,1.00)][P2:(2.00,2.00)], A1:[P3:(4.00,4.00)])
            → tripJoin = A1:[P1:(1.00,1.00)][P2:(2.00,2.00)][P3:(4.00,4.00)]
airTripJoin((A1:[P1:(1.00,1.00)][P2:(2.00,2.00)], A2:[P3:(4.00,4.00)])
            → tripJoin = A1-A2:[P1:(1.00,1.00)][P2:(2.00,2.00)][P3:(4.00,4.00)]
airTripJoin(A1:[P1:(1.00,1.00)][P2:(2.00,2.00)], A2:)
            → tripJoin = A1-A2:[P1:(1.00,1.00)][P2:(2.00,2.00)]
```

- `void airTripDelLast(struct airTrip* trip)`
  Borra la última parada de un recorrido. Para esto debe liberar la memoria, tanto del nodo como del *string* del nombre.

  Ejemplo

  `airTripDelLast(A1:[P1:(1.00,1.00)][P2:(2.00,2.00)])` → `A1:[P1:(1.00,1.00)]`

  `airTripDelLast(A1:)` → `A1:`

- `void airTripRemoveDuplicates(struct airTrip* trip)`
  Borra todas las paradas duplicadas dentro de un recorrido, dejando solo la primer aparición de cada una. *Ejemplo*

```
airTripRemoveDuplicates(A1:[P1:(1.00,1.00)][P1:(2.00,2.00)][P3:(4.00,4.00)])
            → A1:[P1:(1.00,1.00)][P3:(4.00,4.00)]
airTripRemoveDuplicates(A1:[P1:(1.00,1.00)][P2:(2.00,2.00)][P2:(4.00,4.00)])
            → A1:[P1:(1.00,1.00)][P2:(2.00,2.00)]
airTripRemoveDuplicates(A1:[P1:(1.00,1.00)][P2:(2.00,2.00)][P3:(4.00,4.00)])
            → A1:[P1:(1.00,1.00)][P2:(2.00,2.00)][P3:(4.00,4.00)]
```

- `char* airTripGetTrip(struct airTrip* trip)`
  Retorna una *string* con los nombres de todas las paradas del recorrido separadas por un guión. Para el ejemplo de la figura retornaría: `"A-B-C-D-E-F"`.

- `void airTripDelete(struct airTrip* trip)`
  Borra todos los datos del recorrido pasado por parámetro. Debe para esto llamar a la función `free` para cada una de las estructuras y *strings* referenciadas.

Tener presente que todas las funciones que modifiquen las paradas deben también alterar el campo `totalLength` de ser necesario.

Ejercicio 3

A continuación, se enumera un conjunto mínimo de casos de test que deben implementar dentro del archivo `main`:

- `strDup`
  1. String vacío.
  2. String de un carácter.
  3. String que incluya todos los caracteres válidos distintos de cero.

- `strCmp`
  1. Dos string vacíos.
  2. Dos string de un carácter.
  3. Strings iguales hasta un carácter (hacer `cmpStr(s1,s2)` y `cmpStr(s2,s1)`).
  4. Dos strings diferentes (hacer `cmpStr(s1,s2)` y `cmpStr(s2,s1)`).

- `strCnt`
    1. Un *string* vacío y un *string* de 3 caracteres.
    2. Un *string* de 3 caracteres y un *string* vacío.
    3. Dos *strings* de 1 caracter.
    4. Dos *strings* de 5 caracteres.
- `airTripAddLast`
    1. Agregar una parada a un recorrido vacío.
    2. Agregar una parada a un recorrido de un sola parada.
    3. Agregar una parada a un recorrido de más de una parada.
- `airTripAddBest`
    1. Agregar una parada a un recorrido vacío.
    2. Agregar una parada a un recorrido de un sola parada.
    3. Agregar una parada a un recorrido de más de una parada tal que la parada quede en último lugar.
    4. Agregar una parada a un recorrido de más de una parada tal que la parada quede en segundo lugar.
    5. Agregar una parada a un recorrido de más de una parada tal que la parada quede en tercer lugar.
- `airTripJoin`
    1. Unir dos recorridos vacios.
    2. Unir dos recorridos con una sola parada cada uno.
    3. Unir dos recorridos con más de dos paradas cada uno.
    4. Unir un recorrido vacio con otro de una sola parada.
- `airTripDelLast`
    1. Borrar el último de un recorrido de más de dos paradas.
    2. Borrar el último de un recorrido de exactamente dos paradas.
    3. Borrar el último de un recorrido de exactamente una parada.
    4. Borrar el último de un recorrido vacío.
- `airTripRemoveDuplicates`
    1. Borrar duplicados de un recorrido sin duplicados.
    2. Borrar duplicados de un recorrido donde todos son iguales.
    3. Borrar duplicados de un recorrido donde todos se repiten al menos una vez en cualquier orden.
- `airTripGetTrip`
    1. Usar de parámetro un recorrido vacío.
    2. Usar de parámetro un recorrido de solo dos paradas.
    3. Usar de parámetro un recorrido de una sola parada.
- `airTripDelete`
    1. Borrar un recorrido vacío.
    2. Borrar un recorrido de solo una parada.
    3. Borrar un recorrido de más de dos paradas.

Entregable

Para este trabajo práctico no deberán entregar un informe. Sin embargo, deben agregar comentarios en el código que expliquen su solución. No deben comentar qué es lo que hace cada una de las instrucciones sino cuáles son las ideas principales del código implementado y por qué resuelve cada uno de los problemas.
La entrega debe contar con el mismo contenido que fue dado para realizarlo más lo que ustedes hayan agregado, habiendo modificado **solamente** los archivos `airTrip.c` y `main.c`. Es requisito para aprobar entregar el código correctamente comentado.