```cpp
/*
 * File:   main.cpp
 * Author: ANA RONCAL
 * Created on 25 de agosto de 2023, 08:40 PM
 */

#include <cstdlib>
#include <iostream>
#include "ListaD.h"
using namespace std;
#include "funcionesListaDoble.h"

/*
 * LISTA DOBLEMENTE ENLAZADA - TORNEO DE EQUIPOS
 * EL EJERCICIO SE REALIZA CON INSERTAR ORDENADO
 * SIN EMBARGO, SE UTILIZA EL EJERCICIO PARA PROBAR
 * OTRAS FUNCIONES DE LA LISTA DOBLE
 */
int main(int argc, char** argv) {

    struct Lista lista;
    /*CONSTRUIR LA LISTA*/
    construir(lista);

    /*INSERTA AL INICIO DE LA LISTA DOBLE LOS EQUIPOS*/
//    insertarAlInicio(lista, "GutterBall", 9);
//    insertarAlInicio(lista, "KingPins", 8);
//    insertarAlInicio(lista, "PinDoctors", 7);
//    insertarAlInicio(lista, "Scorecards", 10);
//    insertarAlInicio(lista, "Spares", 5);
//    insertarAlInicio(lista, "Splits", 4);
//    insertarAlInicio(lista, "Tenpins", 13);
//    insertarAlInicio(lista, "Woodsplitters", 6);
//    insertarAlInicio(lista, "Lions", 1);
//    insertarAlInicio(lista, "Tigers", 2);

    /*INSERTA AL FINAL DE LA LISTA DOBLE LOS EQUIPOS*/
//    insertarAlFinal(lista, "GutterBall", 9);
//    insertarAlFinal(lista, "KingPins", 8);
//    insertarAlFinal(lista, "PinDoctors", 7);
//    insertarAlFinal(lista, "Scorecards", 10);
//    insertarAlFinal(lista, "Spares", 5);
//    insertarAlFinal(lista, "Splits", 4);
//    insertarAlFinal(lista, "Tenpins", 13);
//    insertarAlFinal(lista, "Woodsplitters", 6);
//    insertarAlFinal(lista, "Lions", 1);
//    insertarAlFinal(lista, "Tigers", 2);

    /*INSERTA EN ORDEN LA LISTA DOBLE LOS EQUIPOS*/
    insertarEnOrden(lista, "GutterBall", 9);
    insertarEnOrden(lista, "KingPins", 8);
    insertarEnOrden(lista, "PinDoctors", 7);
    insertarEnOrden(lista, "Scorecards", 10);
    insertarEnOrden(lista, "Spares", 5);
    insertarEnOrden(lista, "Splits", 4);
    insertarEnOrden(lista, "Tenpins", 13);
    insertarEnOrden(lista, "Woodsplitters", 6);
    insertarEnOrden(lista, "Lions", 1);
    insertarEnOrden(lista, "Tigers", 2);

    /*ELIMINA EL PRIMER ELEMENTO DE LA LISTA*/
    eliminaCabeza(lista);

    /*ELIMINA EL ÚLTIMO ELEMENTO DE LA LISTA*/
    eliminaCola(lista);

    /*ELIMINA EL ELEMENTO CUYO NOMBRE COINCIDE CON EL DE LA LISTA*/
    eliminaNodo(lista, "Tenpins");
    eliminaNodo(lista, "Spares");
```

```
70
71      /*IMPRIME LA LISTA DESDE EL INICIO*/
72      imprime(lista);
73      /*IMPRIME LA LISTA DESDE EL FINAL*/
74      imprimeDesdeFinal(lista);
75      /*IMPRIME LA PRIMERA FECHA DEL TORNEO*/
76      imprimePrimerafecha(lista);
77      /*LIBERA MEMORIA*/
78      destruirLista(lista);
79
80      return 0;
81  }
82
83  /*
84   * File:   Equipo.h
85   * Author: ANA RONCAL
86   * Created on 31 de agosto de 2023, 01:22 PM
87   */
88
89  #ifndef EQUIPO_H
90  #define EQUIPO_H
91
92  struct Equipo{
93      char * nombre; /*Nombre del equipo*/
94      int victorias; /*cantidad de victorias*/
95  };
96
97  #endif /* EQUIPO_H */
98
99  /*
100  * File:   Nodo.h
101  * Author: ANA RONCAL
102  * Created on 25 de agosto de 2023, 08:41 PM
103  */
104
105 #ifndef NODO_H
106 #define NODO_H
107 /*LISTA DOBLEMENTE ENLAZADA*/
108 struct Nodo{
109     struct Equipo equipo;
110     struct Nodo * anterior;   /*apunta a su predecesor*/
111     struct Nodo * siguiente;  /*apunta a su sucesor*/
112 };
113
114 #endif /* NODO_H */
115
116 /*
117  * File:   ListaD.h
118  * Author: ANA RONCAL
119  * Created on 25 de agosto de 2023, 08:41 PM
120  */
121
122 #ifndef LISTAD_H
123 #define LISTAD_H
124 /*LISTA DOBLEMENTE ENLAZADAS*/
125 struct Lista{
126     struct Nodo * cabeza; /*apunta al inicio de la lista*/
127     /*Para tener un fácil acceso a la información de la lista es recomendable
128      usar dos apuntadores*/
129     struct Nodo * cola;   /*apunta al fin de la lista*/
130     int longitud;  /*guarda la longitud de la lista doble*/
131 };
132
133 #endif /* LISTAD_H */
134
135 /*
136  * File:   funcionesListaDoble.h
137  * Author: ANA RONCAL
138  * Created on 25 de agosto de 2023, 08:41 PM
```

```
 */

#ifndef FUNCIONESLISTADOBLE_H
#define FUNCIONESLISTADOBLE_H

void construir(struct Lista & tad);
bool esListaVacia(const struct Lista &);

void imprimePrimerafecha(const struct Lista &);
void imprimeDesdeFinal(const struct Lista &);
void imprime(const struct Lista &);

struct Nodo * obtenerNodoAnterior(const struct Lista & , int);
struct Nodo * crearNodo(const char * , int , struct Nodo * , struct Nodo *);

void insertarAlFinal(struct Lista & , const char * , int );
void insertarEnOrden(struct Lista & , const char *, int);
void insertarAlInicio(struct Lista & , const char *, int );

struct Nodo * seEncuentra(const struct Lista & , const char * );
void eliminaCabeza(struct Lista  & );
void eliminaCola(struct Lista  & );
void eliminaNodo(struct Lista  & , const char * );

void destruirLista(struct Lista & );

#endif /* FUNCIONESLISTADOBLE_H */

/*
 * File:   funcionesListaDoble.cpp
 * Author: ANA RONCAL
 * Created on 27 de agosto de 2023, 02:06 PM
 */

#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstring>
#include "Equipo.h"
#include "Nodo.h"
#include "ListaD.h"
using namespace std;
#include "funcionesListaDoble.h"

void construir(struct Lista & tad){
    tad.cabeza = nullptr;
    tad.cola = nullptr; /* para un mejor manejo de la lista*/
    tad.longitud = 0;
}
/*DETERMINA SI LA LISTA ESTÁ VACÍA*/
bool esListaVacia(const struct Lista & tad){
    return tad.cabeza == nullptr;
}

/*CREA UN NUEVO ELEMENTO CON VALORES INICIALES*/
struct Nodo * crearNodo(const char * nombre, int victorias,
                        struct Nodo * anterior, struct Nodo * siguiente){
    /*Igual a la lista simplemente enlazada*/
    struct Nodo * nuevoNodo = new struct Nodo;

    /*NOMBRE TIENE UNA ASIGNACIÓN DINÁMICA*/
    nuevoNodo->equipo.nombre = new char[strlen(nombre)+1];
    strcpy(nuevoNodo->equipo.nombre, nombre);
    nuevoNodo->equipo.victorias = victorias;
    nuevoNodo->anterior = anterior;   /*Lista doblemente enlazada*/
    nuevoNodo->siguiente = siguiente;
    return nuevoNodo;
}

```

```
208  /*DEVUELVE LA CANTIDAD DE ELEMENTOS DE LA LISTA*/
209  int longitud(struct Lista tad ){
210      return tad.longitud;
211  }
212
213  /*OBTIENE EL ÚLTIMO ELEMENTO DE LA LISTA*/
214  /*PARA LA LISTA QUE HEMOS IMPLEMENTADO NO ES NECESARIA ESTÁ FUNCIÓN
215   *POR CONTAR CON UN PUNTERO A COLA*/
216  struct Nodo * obtenerUltimoNodo(const struct Lista & tad){
217      /*Igual que la lista simplemente enlazada*/
218      struct Nodo * ultimo = nullptr;
219      struct Nodo * recorrido = tad.cabeza;
220      while(recorrido != nullptr){
221          ultimo = recorrido;
222          recorrido = recorrido->siguiente;
223      }
224      return ultimo;
225  }
226
227  /*OBTIENE EL ELEMENTO ANTERIOR SEGÚN UN CRITERIO DE BÚSQUEDA*/
228  struct Nodo * obtenerNodoAnterior(const struct Lista & tad, int victorias){
229      /*Igual que la lista simplemente enlazada*/
230      struct Nodo * anterior = nullptr;
231      struct Nodo * recorrido = tad.cabeza;
232                                                  /*ordena de mayor a menor*/
233      while(recorrido != nullptr and (recorrido->equipo.victorias > victorias)){
234          anterior = recorrido;
235          recorrido = recorrido->siguiente;
236      }
237      return anterior;
238  }
239
240  /*INSERTA UN ELEMENTO AL INICIO DE LA LISTA DOBLE*/
241  void insertarAlInicio(struct Lista & tad, const char * nombre, int victorias){
242      struct Nodo * nuevoNodo = crearNodo(nombre, victorias, nullptr, nullptr);
243      if (tad.cabeza != nullptr)
244          tad.cabeza->anterior = nuevoNodo;
245      nuevoNodo->siguiente = tad.cabeza;
246      tad.cabeza = nuevoNodo;
247      if(nuevoNodo->siguiente == nullptr)
248           tad.cola = nuevoNodo;
249
250      tad.longitud++;
251  }
252
253  /*INSERTA UN ELEMENTO AL FINAL DE LA LISTA DOBLE*/
254  void insertarAlFinal(struct Lista & tad, const char * nombre, int victorias){
255      struct Nodo * nuevoNodo = crearNodo(nombre, victorias, nullptr, nullptr);
256      Nodo * ultimoNodo = tad.cola; /*obtiene el último nodo*/
257      if (ultimoNodo == nullptr){
258          tad.cabeza = nuevoNodo;
259          tad.cola = nuevoNodo;
260      }
261      else{
262          ultimoNodo->siguiente = nuevoNodo;
263          nuevoNodo->anterior = ultimoNodo;
264          tad.cola = nuevoNodo;
265      }
266      tad.longitud++;
267  }
268
269  /*INSERTA EN ORDEN (DE MAYOR A MENOR O VICEVERSA) UN ELEMENTO A LA LISTA*/
270  void insertarEnOrden(struct Lista & tad, const char * nombre, int victorias){
271      /*Diferente a la lista simplemente enlazada*/
272      struct Nodo * nodoAnterior = obtenerNodoAnterior(tad, victorias);
273      struct Nodo * nuevoNodo = crearNodo(nombre, victorias, nullptr, nullptr);
274
275      if (nodoAnterior == nullptr){
276          nuevoNodo->siguiente = tad.cabeza;
```

```
277             if (tad.cabeza != nullptr){ /*lista con datos*/
278                 tad.cabeza->anterior = nuevoNodo;
279                 //tad.cola = nuevoNodo;   /*va en la cola*/
280             }
281             tad.cabeza = nuevoNodo;  /*ambos apuntan a nuevoNodo*/
282             if(tad.cabeza->siguiente == nullptr) /*apunta la primera vez a cabeza*/
283                 tad.cola = nuevoNodo;
284         }else{
285             nuevoNodo->siguiente = nodoAnterior->siguiente;
286             if (nodoAnterior->siguiente != nullptr)
287                 nodoAnterior->siguiente->anterior = nuevoNodo; /*insertar antes del final*/
288             else
289                 tad.cola = nuevoNodo; /*va en la cola*/
290             nodoAnterior->siguiente = nuevoNodo;
291             nuevoNodo->anterior = nodoAnterior;
292         }
293         tad.longitud++;
294     }
295
296     /*MUESTRA LOS EQUIPOS DE LA PRIMERA FECHA DEL TORNEO, UTILIZA LOS PUNTEROS CABEZA
297      Y COLA*/
298     void imprimePrimerafecha(const struct Lista & tad){
299         /*Muestra los resultados de la primera ronda de equipos*/
300         struct Nodo * arriba = tad.cabeza;
301         struct Nodo * abajo = tad.cola;
302
303         cout<<endl<<"PRIMERA FECHA DEL PARTIDO"<<endl<<endl;
304         int max = tad.longitud / 2;
305
306         for(int i = 0; i < max; i++){
307             cout<<"PARTIDO "<<i+1<<endl;
308             cout<<arriba->equipo.nombre<<endl;
309             arriba = arriba->siguiente;
310             cout<<abajo->equipo.nombre<<endl;
311             abajo = abajo->anterior;
312             cout<<endl;
313         }
314     }
315
316     /*Se recorre la lista en forma secuencial*/
317     /*CONST se utiliza para no modificar la estructura, la proteje de ti mismo
318      Si se quiere acceder a los datos, por ejemplo tad.longitud = 20, no te deja
319      dado que la estructura es de lectura solamente con const aquello que pasas
320      no va cambiar, el compilador se da cuenta que es constante*/
321     void imprime(const struct Lista & tad){
322
323         cout<<"IMPRIME DESDE EL INICIO"<<endl;
324         if (esListaVacia(tad)){
325             cout<<"La lista está vacía"<<endl;
326         }
327         else{
328             struct Nodo * recorrido = tad.cabeza;
329             cout<<"EQUIPOS "<<" Victorias"<<endl;
330             while(recorrido != nullptr){
331                 cout<<left<<setw(15)<<recorrido->equipo.nombre;
332                 cout<<right<<setw(3)<<recorrido->equipo.victorias<<endl;
333                 recorrido = recorrido->siguiente;
334             }
335         }
336         cout<<endl;
337     }
338
339     /*IMPRIME DESDE EL FINAL DE LA LISTA DOBLE*/
340     void imprimeDesdeFinal(const struct Lista & tad){
341         /*Se puede hacer uso de la función que obtiene el último nodo*/
342         //struct Nodo * recorrido = obtenerUltimoNodo(tad);
343         /*SIN EMBARGO, VAMOS A UTILIZAR EL PUNTERO COLA*/
344         struct Nodo * recorrido = tad.cola;
345         cout<<"IMPRIME DESDE EL FINAL"<<endl;
```

```
346         cout<<"EQUIPOS "<<" Victorias"<<endl;
347         while(recorrido != nullptr){
348             cout<<left<<setw(15)<<recorrido->equipo.nombre<<"  ";
349             cout<<right<<setw(3)<<recorrido->equipo.victorias<<endl;
350             recorrido = recorrido->anterior;
351         }
352         cout<<endl;
353     }
354 
355     /*LIBERA LA MEMORIA*/
356     void destruirLista(struct Lista & tad){
357         struct Nodo * recorrido = tad.cabeza;
358         struct Nodo * eliminarNodo;
359 
360         while(recorrido != nullptr){
361             eliminarNodo = recorrido;
362             recorrido = recorrido->siguiente;
363             delete eliminarNodo;
364         }
365         tad.cabeza = nullptr;
366         tad.cola = nullptr;
367         tad.longitud = 0;
368     }
369     /*DEVUELVE EL NODO (ELEMENTO) QUE COINCIDE CON LA BÚSQUEDA (NOMBRE)*/
370     struct Nodo * seEncuentra(const struct Lista & tad, const char * nombre){
371         struct Nodo * recorrido = tad.cabeza;
372         /*Mientras sea diferente a nombre que avance*/
373         while(recorrido != nullptr and strcmp(recorrido->equipo.nombre, nombre) != 0){
374             recorrido = recorrido->siguiente;
375         }
376         return recorrido;
377     }
378 
379     /*ELIMINA EL PRIMER ELEMENTO DE LA LISTA*/
380     void eliminaCabeza(struct Lista  & tad){
381         struct Nodo * nodoEliminar = tad.cabeza;
382         if (nodoEliminar == nullptr ){
383             cout<<"No se puede eliminar la cabeza pues la lista está vacía";
384             exit(1);
385         }
386         tad.cabeza = tad.cabeza->siguiente;
387         if(tad.cabeza != nullptr)
388             tad.cabeza->anterior = nullptr;
389         delete nodoEliminar;
390         tad.longitud--;
391     }
392 
393     /*ELIMINA EL ÚLTIMO ELEMENTO DE LA LISTA*/
394     void eliminaCola(struct Lista  & tad){
395         /*COMO SE CUENTA CON UN PUNTERO A COLA, LO USAMOS*/
396         struct Nodo * nodoAEliminar = tad.cola; /* va al final de la lista*/
397          if (nodoAEliminar == nullptr ){
398             cout<<"No se puede eliminar la cola pues la lista está vacía";
399             exit(1);
400         }
401         if(tad.cola != nullptr){
402             tad.cola = nodoAEliminar->anterior;
403             if(tad.cola == nullptr)
404                 tad.cabeza = tad.cola; // se quedó vacía la lista
405             delete nodoAEliminar;
406             if(tad.cola != nullptr)
407                 tad.cola->siguiente = nullptr; /* cambio */
408             tad.longitud--;
409         }
410     }
411 
412     /*ELIMINA EL NODO QUE COINCIDE CON LA BÚSQUEDA*/
413     void eliminaNodo(struct Lista  & tad, const char * nombre){
```

```
414         struct Nodo * ultimo = nullptr;
415         struct Nodo * recorrido = tad.cabeza;
416         while(recorrido != nullptr and strcmp(recorrido->equipo.nombre, nombre) != 0){
417             ultimo = recorrido;
418             recorrido = recorrido->siguiente;
419         }
420         if (recorrido != nullptr){
421             if (ultimo == nullptr){
422                 tad.cabeza = recorrido->siguiente;
423                 if (tad.cabeza != nullptr)
424                     tad.cabeza->anterior = nullptr;
425             }
426             else{
427                 ultimo->siguiente = recorrido->siguiente;
428                 if (recorrido->siguiente != nullptr)
429                     recorrido->siguiente->anterior = ultimo;
430                 if(ultimo->siguiente == nullptr)
431                     tad.cola = ultimo;
432             }
433             delete recorrido;
434             tad.longitud--;
435 
436         }
437     }
```