```
/*
 * File:   main.cpp
 * Author: ANA RONCAL
 * Created on 18 de septiembre de 2023, 05:39 PM
 */

#include <iostream>
#include <iomanip>
#include <cstdlib>
#include "ListaArista.h"
#include "ListaVertice.h" //Necesita
#include "Grafo.h"
using namespace std;
#include "funcionesListaVertice.h"
#include "funcionesGrafo.h"
#include "NodoListaVertice.h"

/*
 * IMPLEMENTA GRAFOS
 */
int main(int argc, char** argv) {

    struct Grafo grafo;

    //Construir el Grafo
    construirGrafo(grafo);

    //Agregar los vértices
    agregarVertice(grafo, 'A');
    agregarVertice(grafo, 'B');
    agregarVertice(grafo, 'C');
    agregarVertice(grafo, 'D');

    //Eliminar el vértice
    //eliminarVertice(grafo, 'B');

    //Agregar las aristas
    agregarArista(grafo, 'A', 'B');
    agregarArista(grafo, 'C', 'A');
    agregarArista(grafo, 'D', 'A');
    agregarArista(grafo, 'D', 'C');

    //eliminar la arista
    //eliminarArista(grafo, 'D', 'C');

    //Muestra los resultados
    destruirGrafo(grafo);
    mostrarVerticeYAristas(grafo);
   return 0;
}

/*
 * File:   Grafo.h
 * Author: ANA RONCAL
 * Created on 27 de septiembre de 2023, 09:17 AM
 */

#ifndef GRAFO_H
#define GRAFO_H

struct Grafo{
    struct ListaVertice listaVertice;
    int longitud;
};

#endif /* GRAFO_H */

/*
 * File:   NodoListaArista.h
```

```
 * Author: ANA RONCAL
 * Created on 27 de septiembre de 2023, 09:25 AM
 */

#ifndef NODOLISTAARISTA_H
#define NODOLISTAARISTA_H

struct NodoListaArista {
    char elemento;    //ElementoListaArista: ACÁ PUEDE CAMBIAR EL ELEMENTO
    struct NodoListaArista * siguiente;
} ;


#endif /* NODOLISTAARISTA_H */

/*
 * File:   NodoListavertice.h
 * Author: ANA RONCAL
 * Created on 27 de septiembre de 2023, 09:22 AM
 */

#ifndef NODOLISTAVERTICE_H
#define NODOLISTAVERTICE_H

struct NodoListaVertice{
    char elemento;    //ACÄ puede cambiar el elemento
    struct NodoListaVertice * siguiente;
    struct ListaArista listaArista;
};

#endif /* NODOLISTAVERTICE_H */

/*
 * File:   ListaArista.h
 * Author: ANA RONCAL
 * Created on 27 de septiembre de 2023, 09:25 AM
 */

#ifndef LISTAARISTA_H
#define LISTAARISTA_H

struct ListaArista{
    int longitud;
    struct NodoListaArista * cabeza;
};

#endif /* LISTAARISTA_H */

/*
 * File:   ListaVertica.h
 * Author: ANA RONCAL
 * Created on 27 de septiembre de 2023, 09:20 AM
 */

#ifndef LISTAVERTICA_H
#define LISTAVERTICA_H

struct ListaVertice{
    struct NodoListaVertice  * cabeza;
    int longitud;
};

#endif /* LISTAVERTICA_H */

/*
 * File:   funcionesListaArista.h
 * Author: ANA RONCAL
 * Created on 27 de septiembre de 2023, 11:01 AM
 */
```

```
139
140 #ifndef FUNCIONESLISTAARISTA_H
141 #define FUNCIONESLISTAARISTA_H
142
143 void construirListaAristas(struct ListaArista & listaArista);
144 bool seEncuentraAristaEnListaArista(struct ListaArista listaArista, char llave);
145 struct NodoListaArista * obtenerUltimoNodoA( struct ListaArista  listaArista);
146 struct NodoListaArista * crearNodoA(char elemento, struct NodoListaArista * siguiente);
147 void insertarAristaAlFinal(struct ListaArista & listaArista, char vertice);
148
149 void eliminarAristaEnLista(struct ListaArista & listaArista, char vertice);
150 void destruirListaArista(struct ListaArista & listaArista);
151
152 #endif /* FUNCIONESLISTAARISTA_H */
153
154 /*
155  * File:   funcionesListaVertice.h
156  * Author: ANA RONCAL
157  * Created on 27 de septiembre de 2023, 09:38 AM
158  */
159
160 #ifndef FUNCIONESLISTAVERTICE_H
161 #define FUNCIONESLISTAVERTICE_H
162
163 void construirListaVertice(struct ListaVertice & listaVertice);
164 bool esListaVerticeVacio(struct ListaVertice listaVercice);
165 bool seEncuentraVerticeLista(struct ListaVertice listaVercice, char elemento);
166 struct NodoListaVertice * crearNodoV(char elemento, struct NodoListaVertice * siguiente);
167 struct NodoListaVertice * obtenerUltimoNodoV( struct ListaVertice listaVertice);
168 void insertarVerticeAlFinal(struct ListaVertice & listaVertice, char element);
169 bool seEncuentraAristaLista(struct ListaVertice listaVertice, char verticeOrigen, char
    verticeDestino);
170 struct NodoListaVertice * obtenerNodoVertice(struct ListaVertice listaVertice, char clave
    );
171 void insertarListaAristas(struct ListaVertice &, char verticeOrigen, char verticeDestino
    );
172
173
174 void eliminarVerticeEnLista(struct ListaVertice & listaVertice, char vertice);
175 void eliminarDeListaDeAristas(struct ListaVertice & listaVertice, char verticeOrigen,
    char verticeDestino);
176 void destruirListaVertice(struct ListaVertice & listaVertice);
177 #endif /* FUNCIONESLISTAVERTICE_H */
178
179 /*
180  * File:   funcionesGrafo.h
181  * Author: ANA RONCAL
182  * Created on 27 de septiembre de 2023, 09:19 AM
183  */
184
185 #ifndef FUNCIONESGRAFO_H
186 #define FUNCIONESGRAFO_H
187 void construirGrafo(struct Grafo & grafo);
188 void agregarVertice(struct Grafo & grafo, char);
189 bool esGrafoVacio(struct Grafo grafo);
190 void agregarArista(struct Grafo & grafo, char vertice1, char vertice2);
191 void eliminarVertice(struct Grafo & grafo, char vertice);
192 int longitudGrafo(Grafo);
193
194 void mostrarVerticeYAristas(struct Grafo grafo);
195 int seEncuentraVerticeOrigen(struct Grafo grafo, char vertice);
196 void eliminarArista(struct Grafo & grafo, char verticeOrigen, char verticeDestino);
197 void destruirGrafo(struct Grafo & grafo);
198 #endif /* FUNCIONESGRAFO_H */
199
200 /*
201  * File:   funcionesListaArista.cpp
202  * Author: ANA RONCAL
203  * Created on 27 de septiembre de 2023, 11:02 AM
```

```
 */

#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstring>
#include "ListaArista.h"
#include "ListaVertice.h"
#include "NodoListaArista.h"

using namespace std;
#include "funcionesListaArista.h"

void construirListaAristas(struct ListaArista & listaArista){
    listaArista.cabeza = nullptr;
    listaArista.longitud = 0;
}

bool seEncuentraAristaEnListaArista(struct ListaArista listaArista, char llave){
    struct NodoListaArista * recorrido = listaArista.cabeza;
    while((recorrido != nullptr) and (recorrido->elemento != llave)){
        recorrido = recorrido->siguiente;
    }
    return recorrido != nullptr;
}

struct NodoListaArista * crearNodoA(char elemento, struct NodoListaArista * siguiente){
    struct NodoListaArista * nuevoNodo = new struct NodoListaArista;
    nuevoNodo->elemento = elemento;
    nuevoNodo->siguiente = siguiente;
    return nuevoNodo;
}

//Notar que esta función retorna nulo cuando la lista es vacía
/*Obtiene el último nodo de la lista*/
struct NodoListaArista * obtenerUltimoNodoA(struct ListaArista  listaArista){
    struct NodoListaArista * ultimo = nullptr;
    struct NodoListaArista * recorrido = listaArista.cabeza;

    while(recorrido != nullptr){
        ultimo = recorrido;
        recorrido = recorrido->siguiente;
    }
    return ultimo;
}

void insertarAristaAlFinal(struct ListaArista & listaArista, char vertice){
    struct NodoListaArista * ultimoNodo = obtenerUltimoNodoA(listaArista);
    struct NodoListaArista * nuevoNodo = crearNodoA(vertice, nullptr);

    if (ultimoNodo == nullptr)  /*Si la lista está vacia*/
        listaArista.cabeza = nuevoNodo; /*se inserta en la cabeza de la lista*/
    else    //La lista ya tiene nodos
        ultimoNodo->siguiente = nuevoNodo;
    listaArista.longitud++;
}

void eliminarAristaEnLista(struct ListaArista & listaArista, char vertice){
    struct NodoListaArista * ultimo = nullptr;
    struct NodoListaArista * recorrido = listaArista.cabeza;
    while(recorrido != nullptr and recorrido->elemento != vertice){
        ultimo = recorrido;
        recorrido = recorrido->siguiente;
    }
    if (recorrido != nullptr){
        if (ultimo == nullptr)
            listaArista.cabeza = recorrido->siguiente;
        else
            ultimo->siguiente = recorrido->siguiente;
```

```
273             delete recorrido;
274             listaArista.longitud--;
275         }
276     }
277 
278     void destruirListaArista(struct ListaArista & listaArista){
279         struct NodoListaArista * recorrido = listaArista.cabeza;
280         while(recorrido != nullptr){
281             struct NodoListaArista * nodoAEliminar = recorrido;
282             recorrido = recorrido->siguiente;
283             delete(nodoAEliminar);
284         }
285         listaArista.cabeza = nullptr;
286         listaArista.longitud = 0;
287     }
288 
289     /*
290      * File:   funcionesListavertice.cpp
291      * Author: ANA RONCAL
292      * Created on 27 de septiembre de 2023, 09:39 AM
293      */
294 
295     #include <iostream>
296     #include <iomanip>
297     #include <fstream>
298     #include <cstring>
299     #include "ListaArista.h" //necesita
300     #include "ListaVertice.h"
301     #include "NodoListaVertice.h"
302     using namespace std;
303     #include "funcionesListaArista.h"
304     #include "funcionesListaVertice.h"
305 
306     void construirListaVertice(struct ListaVertice & listaVertice){
307         listaVertice.cabeza = nullptr;
308         listaVertice.longitud = 0;
309     }
310 
311     bool esListaVerticeVacio(struct ListaVertice listaVertice){
312         return listaVertice.cabeza == nullptr;
313     }
314 
315     bool seEncuentraVerticeLista(struct ListaVertice listaVertice, char llave){
316         struct NodoListaVertice * recorrido = listaVertice.cabeza;
317         while(recorrido != nullptr and recorrido->elemento != llave){
318             recorrido = recorrido->siguiente;
319         }
320         return recorrido != nullptr;
321     }
322 
323     struct NodoListaVertice * crearNodoV(char elemento, struct NodoListaVertice * siguiente){
324         struct NodoListaVertice * nuevoNodo = new struct NodoListaVertice;
325 
326         nuevoNodo->elemento = elemento;
327         nuevoNodo->siguiente = siguiente;
328         construirListaAristas(nuevoNodo->listaArista);
329         return nuevoNodo;
330     }
331 
332     //Notar que esta función retorna nulo cuando la lista es vacía
333     /*Obtiene el último nodo de la lista*/
334     struct NodoListaVertice * obtenerUltimoNodoV( struct ListaVertice  listaVertice){
335         struct NodoListaVertice * ultimo = nullptr;
336         struct NodoListaVertice * recorrido = listaVertice.cabeza;
337 
338         while(recorrido != nullptr){
339             ultimo = recorrido;
340             recorrido = recorrido->siguiente;
341         }
```

```
342         return ultimo;
343     }
344 
345     void insertarVerticeAlFinal(struct ListaVertice & listaVertice, char elemento){
346         struct NodoListaVertice * ultimoNodo = obtenerUltimoNodoV(listaVertice);
347         struct NodoListaVertice * nuevoNodo = crearNodoV(elemento, nullptr);
348 
349         if (ultimoNodo == nullptr)  /*Si la lista está vacia*/
350             listaVertice.cabeza = nuevoNodo; /*se inserta en la cabeza de la lista*/
351         else    //La lista ya tiene nodos
352             ultimoNodo->siguiente = nuevoNodo;
353         listaVertice.longitud++;
354     }
355 
356     struct NodoListaVertice * obtenerNodoVertice(struct ListaVertice listaVertice, char clave
    ){
357         struct NodoListaVertice * recorrido = listaVertice.cabeza;
358         while((recorrido != nullptr) and (recorrido->elemento != clave)){
359             recorrido = recorrido->siguiente;
360         }
361         return recorrido;
362     }
363 
364     bool seEncuentraAristaLista(struct ListaVertice listaVertice, char verticeOrigen, char
    verticeDestino){
365         struct NodoListaVertice * nodoVerticeOrigen = obtenerNodoVertice(listaVertice,
        verticeOrigen);
366         if (nodoVerticeOrigen == nullptr)
367         return false;
368         return seEncuentraAristaEnListaArista(nodoVerticeOrigen->listaArista, verticeDestino
        );
369     }
370 
371     void insertarListaAristas(struct ListaVertice & listaVertice, char verticeOrigen, char
    verticeDestino){
372         struct NodoListaVertice * nodoVerticeOrigen = obtenerNodoVertice(listaVertice,
        verticeOrigen);
373         if (nodoVerticeOrigen == nullptr){
374             cout<<"Error al insertar la lista de arista. No se ha encontrado el vértice"<<
            verticeOrigen<<endl;
375             return;
376         }
377 
378         if ( not seEncuentraAristaEnListaArista(nodoVerticeOrigen->listaArista,
        verticeDestino)){
379             insertarAristaAlFinal(nodoVerticeOrigen->listaArista, verticeDestino);
380         }
381     }
382 
383     void eliminarDeListaDeAristas(struct ListaVertice & listaVertice, char verticeOrigen,
    char verticeDestino){
384         struct NodoListaVertice * nodoVerticeOrigen = obtenerNodoVertice(listaVertice,
        verticeOrigen);
385         if (nodoVerticeOrigen != nullptr)
386             eliminarAristaEnLista(nodoVerticeOrigen->listaArista, verticeDestino);
387 
388     }
389 
390     void eliminarVerticeEnLista(struct ListaVertice & listaVertice, char vertice){
391         struct NodoListaVertice * ultimoNodo = nullptr;
392         struct NodoListaVertice * recorrido = listaVertice.cabeza;
393 
394         while(recorrido != nullptr and recorrido->elemento != vertice){
395             ultimoNodo = recorrido;
396             recorrido = recorrido->siguiente;
397         }
398         if (recorrido != nullptr){
399             if (ultimoNodo == nullptr)
400                 listaVertice.cabeza=recorrido->siguiente;
```

```
401             else
402                 ultimoNodo->siguiente = recorrido->siguiente;
403             destruirListaArista(recorrido->listaArista);
404             free(recorrido);
405             listaVertice.longitud--;
406         }
407     }
408 
409     void destruirListaVertice(struct ListaVertice & listaVertice){
410         struct NodoListaVertice * recorrido = listaVertice.cabeza;
411         while(recorrido != nullptr){
412             destruirListaArista(recorrido->listaArista);
413             struct NodoListaVertice * nodoAEliminar = recorrido;
414             recorrido = recorrido->siguiente;
415             delete nodoAEliminar;
416         }
417         listaVertice.cabeza = nullptr;
418         listaVertice.longitud = 0;
419     }
420 
421     /*
422      * File:   funcionesGrafo.cpp
423      * Author: ANA RONCAL
424      * Created on 27 de septiembre de 2023, 09:36 AM
425      */
426 
427     #include <iostream>
428     #include <iomanip>
429     #include <fstream>
430     #include <cstring>
431     #include "NodoListaArista.h"
432     #include "ListaArista.h"
433     #include "NodoListaVertice.h"
434     #include "ListaVertice.h"
435     #include "Grafo.h"
436 
437     using namespace std;
438     #include "funcionesListaArista.h"
439     #include "funcionesListaVertice.h"
440     #include "funcionesGrafo.h"
441 
442     void construirGrafo(struct Grafo & grafo){
443         construirListaVertice(grafo.listaVertice);
444         grafo.longitud = 0;
445     }
446 
447     bool esGrafoVacio(struct Grafo grafo){
448         return esListaVerticeVacio(grafo.listaVertice);
449     }
450 
451     bool seEncuentraVertice(struct Grafo grafo, int elemento){
452         return seEncuentraVerticeLista(grafo.listaVertice, elemento);
453     }
454 
455     /*Agregar añade un vértice, este puede ser de cualquier tipo de dato*/
456     /*es decir elemento está representando un tipo VERTICE*/
457     void agregarVertice(struct Grafo & grafo, char elemento){
458         if(not seEncuentraVertice(grafo, elemento)){
459             insertarVerticeAlFinal(grafo.listaVertice, elemento);
460             grafo.longitud++;
461         }
462     }
463 
464     bool seEncuentraArista(struct Grafo grafo, char verticeOrigen, char verticeDestino){
465         return seEncuentraAristaLista(grafo.listaVertice, verticeOrigen, verticeDestino);
466     }
467 
468     void agregarArista(struct Grafo & grafo, char verticeOrigen, char verticeDestino){
469         bool seEncuentraVerticeOrigen = seEncuentraVertice(grafo, verticeOrigen);
```

```
    bool seEncuentraVerticeDestino = seEncuentraVertice(grafo, verticeDestino);
    if((not seEncuentraVerticeOrigen) or (not seEncuentraVerticeDestino)){
        cout<<"No se ha encontrado algún vértice";
        return;
    }
    if(not seEncuentraArista(grafo, verticeOrigen, verticeDestino)){
        insertarListaAristas(grafo.listaVertice, verticeOrigen, verticeDestino);
    }
}

int longitudGrafo(struct Grafo grafo){
    return grafo.longitud;
}

void eliminarVertice(struct Grafo & grafo, char vertice){
    if (seEncuentraVertice(grafo, vertice)){
        eliminarVerticeEnLista(grafo.listaVertice, vertice);
        grafo.longitud--;
    }
}

void eliminarArista(struct Grafo & grafo, char verticeOrigen, char verticeDestino){
    int seEncuentraVertOrigen = seEncuentraVertice(grafo, verticeOrigen);
    if (seEncuentraVertOrigen)
        eliminarDeListaDeAristas(grafo.listaVertice, verticeOrigen, verticeDestino);

}

void mostrarVerticeYAristas(struct Grafo grafo){
    struct NodoListaVertice * recorridoG = new struct NodoListaVertice;
    struct NodoListaArista * recorridoA = new struct NodoListaArista;
    recorridoG = grafo.listaVertice.cabeza;

    if (recorridoG == nullptr){
        cout<<"No se puede mostrar, el grafo está vacío"<<endl;
        return;
    }
    else{
        while(recorridoG != nullptr){
            cout<<setw(5)<<recorridoG->elemento<<":";
            recorridoA = recorridoG->listaArista.cabeza;
            while(recorridoA != nullptr){
                cout<<setw(5)<<recorridoA->elemento;
                recorridoA = recorridoA->siguiente;
            }
            recorridoG = recorridoG->siguiente;
            cout<<endl;
        }
        delete recorridoG;
    }
}

void destruirGrafo(struct Grafo & grafo){
    destruirListaVertice(grafo.listaVertice);

    grafo.longitud = 0;
}
```