```cpp
#ifndef GRAFICAS_4_H_INCLUDED
#define GRAFICAS_4_H_INCLUDED

#include <iostream>
#define VACIO 99999

struct caja2;

struct caja1{
    int numNodo;
    float longitud;
    caja1 *siguiente;
    caja2 *direccionNodo;
};

class lista_arcos{
    caja1 *principio, *anterior, *lugarAgregado;
    int encontrado, donde, cuantos;
    enum encontrado{SI, NO};
    enum donde{PRINCIPIO, ENMEDIO, FINAL};

public:
    lista_arcos();
    ~lista_arcos();
    void buscar(int a);
    int agregar(int a);
    int borrar(int a);
    void pintar();
    caja1 *lugar_agregado();
    caja1 *Principio();
    void iniciar();
    void terminar();
};

//-------------------------------------------------------------------------------------------------------------------------------

struct caja3;

struct caja2{
    int numNodo;
    int bandera;
    float rutaCorta;
    caja2 *antecesor;
    caja3 *direccion;
    lista_arcos salientes, entrantes;
    caja2 *siguiente;
};

/*******************************************************************************************************************************
*******************************************************************************/

lista_arcos::lista_arcos(){
    principio = anterior = lugarAgregado = NULL;
    encontrado = NO;
    donde = VACIO;
    cuantos = 0;
}

//-------------------------------------------------------------------------------------------------------------------------------

void lista_arcos::iniciar(){
    principio = anterior = lugarAgregado = NULL;
    encontrado = NO;
    donde = VACIO;
    cuantos = 0;
```

```
 61  }
 62
//-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
 63
 64  void lista_arcos::buscar(int a){
 65      caja1 *p; //Puntero tipo caja que recorrerá la estructura.
 66
 67      //Si la estructura está vacía entonces se indica que el dato no se encontró y que la lista está vacia.
 68      if(principio == NULL){
 69          encontrado = NO;
 70          donde = VACIO;
 71          return;
 72      }
 73
 74      //Si la lista no está vacía, entonces se comenzará a recorrer desde el principio hasta el final.
 75      p = principio;
 76      while(p){
 77
 78          //Si se encuentra el dato, entonces la variable encontrado toma el valor de SI.
 79          if(p -> numNodo == a){
 80              encontrado = SI;
 81              lugarAgregado = p;
 82              if(p == principio){ //Si se encuentra al principio, entonces donde = PRINCIPIO.
 83                  donde = PRINCIPIO;
 84              }else if(p -> siguiente == NULL){ //Si la siguiente dirección es NULL, entonces significa que
estamos en el final de la lista.
 85                  donde = FINAL;
 86               }
 87               else{ //Si no estamos al principio ni al final de la lista ordenada, entonces encontramos el
número enmedio de la estructura.
 88                  donde = ENMEDIO;
 89              }
 90              return;
 91          }
 92          /*
 93          Como estamos ordenando números de menor a mayor, entonces si encontramos algún número menor a 'a',
simplemente seguimos recorriendo
 94          la estructura.
 95          */
 96          else if(p -> numNodo < a){
 97              encontrado = NO; //Se indica que no se ha encontrado aún.
 98              anterior = p; //El puntero '*anterior' toma el valor de la dirección en donde nos encontramos.
 99              p = p -> siguiente; //Finalmente p se pone sobre la siguiente caja de la lista ordenada.
100          }
101          /*
102          Como estamos ordenando números de menor a mayor, entonces, si encontramos algún número mayor a 'a',
significa que nuestro nuevo
103          valor debe agregarse una posición antes de este número mayor con el que nos encontramos.
104          */
105          else{
106              encontrado = NO; //Se indica que 'a' no ha sido encontrado.
107              if(principio == p){ //Si el primer elemento que checamos resultó ser mayor, entonces significa
que 'a' debe agregarse al principio.
108                  donde = PRINCIPIO;
109              }
110              else{ //Si no lo encontramos al principio, entonces forzosamente debe de agregarse en algún
lugar enmedio de la estructura.
111                  donde = ENMEDIO;
112              }
113              return;
114          }
115      }
116      /*
117      Si el ciclo se acaba y se recorre toda la lista sin encontrar ningún número mayor, entonces significa
que el número que queremos
```

```
118      encontrar es mayor a todos los que existen dentro de la lista ordenada; por lo tanto, no ha sido
encontrado y debe agregarse al final.
119      */
120      encontrado = NO;
121      donde = FINAL;
122      return;
123  }
124
//-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
125
126  int lista_arcos::agregar(int a){
127      caja1 *p; //Puntero tipo caja que nos servirá para agregar el valor a la estructura.
128      buscar(a); //Se busca el número que queremos introducir.
129
130      //Como no se consideran repeticiones, si el número se encontró, se regresa un cero porque no volverá a
ser introducido a la estructura.
131      if(encontrado == SI) return (0);
132
133      //Si no se encontró, entonces se procede a agregar al número 'a'.
134      p = new caja1; //Se crea una nueva caja.
135      p -> numNodo = a; //Se introduce el valor 'a' a la caja.
136      lugarAgregado = p;
137
138      /*
139      Si la lista ordenada está vacia, entonces el puntero '*siguiente' de la caja será NULL (porque después
de esta cajita no existirá
140      otra) y el puntero '*principio' tomará la dirección de esta caja que hemos agregado.
141      */
142      if(donde == VACIO){
143          p -> siguiente = NULL;
144          principio = p;
145      }
146      /*
147      Si se debe agregar al principio, entonces el puntero '*siguiente' de la caja apuntará hacia el valor
que, hasta el momento,
148      estaba al principio de la estructura; como hemos agregadoa un valor antes del primer elemento de la
lista, entonces el puntero
149      '*principio' se recorre hacia atrás y tomará la dirección del elemento que recién hemos agregado.
150      */
151      else if(donde == PRINCIPIO){
152          p -> siguiente = principio;
153          principio = p;
154      }
155      /*
156      Si debemos agregar al final, entonces el puntero '*siguiente' de nuestra caja será NULL (pues no
existirá nada después). En este
157      caso, el puntero '*anterior' se encontrará sobre el último elemento de la lista hasta este momento;
entonces, el puntero
158      'anterio -> siguiente' tomara el valor de p porque ahora '*p' será la última cajita de la estructura.
159      */
160      else if(donde == FINAL){
161          p -> siguiente = NULL;
162          anterior -> siguiente = p;
163      }
164      /*
165      Si no se agregará al principio ni al final, entonces se agregará enmedio; para esto los puntero
166      se "puentean" de la siguiente manera:
167      */
168      else{
169          p -> siguiente = anterior -> siguiente;
170          anterior -> siguiente = p;
171      }
172
173      cuantos ++; //Se incrementa el número de datos dentro de la estructura
174      return (1); //Se regresa un 1 porque el valor pudo ser agregado.
```

```
175 }
176
//-----------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
177
178 int lista_arcos::borrar(int a){
179     caja1 *p; //Puntero tipo caja que nos ayudará a recorrer la lista.
180     buscar(a); //Se busca el número.
181
182     if(encontrado == NO) return (0); //Si no se encuentra, entonces no puede eliminarse y se regresa un 0.
183
184     /*
185     Si se encuentra al principio, entonces '*p' se posiciona sobre el puntero '*principio'; entonces, como
el primer
186     elemento se va a eliminar, el número siguiente pasará a ser el primero; por lo tanto principio = p ->
siguiente.
187     */
188     if(donde == PRINCIPIO){
189         p = principio;
190         principio = p -> siguiente;
191     }
192     /*
193     Si se encuentra en medio, entonces utilizamos al puntero '*anterior' para poner a '*p' sobre el
elemento que queremos
194     eliminar; después, se "puentea" el apuntador '*siguiente' de la caja con dirección '*anterior'
195     */
196     else if(donde == ENMEDIO){
197         p = anterior -> siguiente;
198         anterior -> siguiente = p -> siguiente;
199     }
200     /*
201     Si no se encontró al principio ni enmedio, entonces se encontró al final. Utilizamos el puntero
'*anterior' para posicionarnos sobre
202     el último elemento; después, 'anterior -> siguiente' será NULL, porque después del elemento '*anterior'
ahora ya no habrá nada.
203     */
204     else{
205         p = anterior -> siguiente;
206         anterior -> siguiente = NULL;
207     }
208
209     delete(p); //Se elimina el dato deseado.
210     cuantos--; //Se decrementa el número de datos dentro de la lista.
211     return (1); //Se regresa un 1 porque sí pudimos eliminar el dato.
212 }
213
//-----------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
214
215 void lista_arcos::pintar(){
216     caja1 *p; p = principio; //Se crea un puntero tipo caja y se sitúa al principio de la lista ordenada.
217     while(p){ //Se imprimen todos los valores que forman parte de la estructura.
218         std::cout << p -> numNodo << "-" << p->direccionNodo->numNodo << " Longitud: " << p->longitud << "
; ";
219         p = p -> siguiente;
220     }
221     std::cout << "\b\b ";
222 }
223
//-----------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
224
225 caja1* lista_arcos::lugar_agregado(){
226     return(lugarAgregado);
227 }
228
```

```
//-------------------------------------------------------------------------------------------------------------------------------
229
230 caja1 *lista_arcos::Principio(){
231     return (principio);
232 }
233
234
//-------------------------------------------------------------------------------------------------------------------------------
235 lista_arcos::~lista_arcos(){
236     caja1 *p;
237     while(principio){
238         p = principio;
239         principio = p -> siguiente;
240         delete p;
241     }
242     principio = anterior = lugarAgregado = NULL;
243     encontrado = NO;
244     donde = VACIO;
245     cuantos = 0;
246     return;
247 }
248
//-------------------------------------------------------------------------------------------------------------------------------
249
250
251
252
//-------------------------------------------------------------------------------------------------------------------------------
253 void lista_arcos::terminar(){
254     caja1 *p;
255     while(principio){
256         p = principio;
257         principio = p -> siguiente;
258         delete p;
259     }
260     principio = anterior = lugarAgregado = NULL;
261     encontrado = NO;
262     donde = VACIO;
263     cuantos = 0;
264     return;
265 }
266
267
/*******************************************************************************************************************************
*******************************************************************************/
268
269 class lista_nodos{
270     caja2 *principio, *anterior, *lugarAgregado;
271     int encontrado, donde, cuantos;
272     enum encontrado {SI, NO};
273     enum donde{PRINCIPIO, ENMEDIO, FINAL};
274
275 public:
276     lista_nodos();
277     ~lista_nodos();
278     void buscar(int a);
279     int agregar(int a);
280     int borrar(int a);
281     void pintar();
282     void iniciar();
283     void terminar();
284     caja2* lugar_agregado();
```

```
285 };
286
//-----------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
287
288 lista_nodos::lista_nodos(){
289     principio = anterior = lugarAgregado = NULL;
290     encontrado = NO;
291     donde = VACIO;
292     cuantos = 0;
293 }
294
//-----------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
295
296 void lista_nodos::iniciar(){
297     principio = anterior = lugarAgregado = NULL;
298     encontrado = NO;
299     donde = VACIO;
300     cuantos = 0;
301 }
302
//-----------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
303
304 void lista_nodos::buscar(int a){
305     caja2 *p; //Puntero tipo caja que recorrerá la estructura.
306
307     //Si la estructura está vacía entonces se indica que el dato no se encontró y que la lista está vacia.
308     if(principio == NULL){
309         encontrado = NO;
310         donde = VACIO;
311         return;
312     }
313
314     //Si la lista no está vacía, entonces se comenzará a recorrer desde el principio hasta el final.
315     p = principio;
316     while(p){
317
318         //Si se encuentra el dato, entonces la variable encontrado toma el valor de SI.
319         if(p -> numNodo == a){
320             encontrado = SI;
321             lugarAgregado = p;
322             if(p == principio){ //Si se encuentra al principio, entonces donde = PRINCIPIO.
323                 donde = PRINCIPIO;
324             }else if(p -> siguiente == NULL){ //Si la siguiente dirección es NULL, entonces significa que
estamos en el final de la lista.
325                 donde = FINAL;
326              }
327              else{ //Si no estamos al principio ni al final de la lista ordenada, entonces encontramos el
número enmedio de la estructura.
328                 donde = ENMEDIO;
329             }
330             return;
331         }
332         /*
333         Como estamos ordenando números de menor a mayor, entonces si encontramos algún número menor a 'a',
simplemente seguimos recorriendo
334         la estructura.
335         */
336         else if(p -> numNodo < a){
337             encontrado = NO; //Se indica que no se ha encontrado aún.
338             anterior = p; //El puntero '*anterior' toma el valor de la dirección en donde nos encontramos.
339             p = p -> siguiente; //Finalmente p se pone sobre la siguiente caja de la lista ordenada.
340         }
341         /*
```

```
342          Como estamos ordenando números de menor a mayor, entonces, si encontramos algún número mayor a 'a',
significa que nuestro nuevo
343          valor debe agregarse una posición antes de este número mayor con el que nos encontramos.
344          */
345          else{
346              encontrado = NO; //Se indica que 'a' no ha sido encontrado.
347              if(principio == p){ //Si el primer elemento que checamos resultó ser mayor, entonces significa
que 'a' debe agregarse al principio.
348                  donde = PRINCIPIO;
349              }
350              else{ //Si no lo encontramos al principio, entonces forzosamente debe de agregarse en algún
lugar enmedio de la estructura.
351                  donde = ENMEDIO;
352              }
353              return;
354          }
355      }
356      /*
357      Si el ciclo se acaba y se recorre toda la lista sin encontrar ningún número mayor, entonces significa
que el número que queremos
358      encontrar es mayor a todos los que existen dentro de la lista ordenada; por lo tanto, no ha sido
encontrado y debe agregarse al final.
359      */
360      encontrado = NO;
361      donde = FINAL;
362      return;
363  }
364
//-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
365
366  int lista_nodos::agregar(int a){
367      caja2 *p; //Puntero tipo caja que nos servirá para agregar el valor a la estructura.
368      buscar(a); //Se busca el número que queremos introducir.
369
370      //Como no se consideran repeticiones, si el número se encontró, se regresa un cero porque no volverá a
ser introducido a la estructura.
371      if(encontrado == SI) return (0);
372
373      //Si no se encontró, entonces se procede a agregar al número 'a'.
374      p = new caja2; //Se crea una nueva caja.
375      p -> numNodo = a; //Se introduce el valor 'a' a la caja.
376      p->bandera = 0;
377      p->rutaCorta = 0;
378      p->antecesor = NULL;
379      p->direccion = NULL;
380      (p -> salientes).iniciar();
381      (p -> entrantes).iniciar();
382      lugarAgregado = p;
383
384      /*
385      Si la lista ordenada está vacia, entonces el puntero '*siguiente' de la caja será NULL (porque después
de esta cajita no existirá
386      otra) y el puntero '*principio' tomará la dirección de esta caja que hemos agregado.
387      */
388      if(donde == VACIO){
389          p -> siguiente = NULL;
390          principio = p;
391      }
392      /*
393      Si se debe agregar al principio, entonces el puntero '*siguiente' de la caja apuntará hacia el valor
que, hasta el momento,
394      estaba al principio de la estructura; como hemos agregadoa un valor antes del primer elemento de la
lista, entonces el puntero
395      '*principio' se recorre hacia atrás y tomará la dirección del elemento que recién hemos agregado.
396      */
```

```
397      else if(donde == PRINCIPIO){
398          p -> siguiente = principio;
399          principio = p;
400      }
401      /*
402      Si debemos agregar al final, entonces el puntero '*siguiente' de nuestra caja será NULL (pues no
existirá nada después). En este
403      caso, el puntero '*anterior' se encontrará sobre el último elemento de la lista hasta este momento;
entonces, el puntero
404      'anterio -> siguiente' tomara el valor de p porque ahora '*p' será la última cajita de la estructura.
405      */
406      else if(donde == FINAL){
407          p -> siguiente = NULL;
408          anterior -> siguiente = p;
409      }
410      /*
411      Si no se agregará al principio ni al final, entonces se agregará enmedio; para esto los puntero
412      se "puentean" de la siguiente manera:
413      */
414      else{
415          p -> siguiente = anterior -> siguiente;
416          anterior -> siguiente = p;
417      }
418
419      cuantos ++; //Se incrementa el número de datos dentro de la estructura
420      return (1); //Se regresa un 1 porque el valor pudo ser agregado.
421  }
422
//-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
423
424  int lista_nodos::borrar(int a){
425      caja2 *p; //Puntero tipo caja que nos ayudará a recorrer la lista.
426      buscar(a); //Se busca el número.
427
428      if(encontrado == NO) return (0); //Si no se encuentra, entonces no puede eliminarse y se regresa un 0.
429
430      /*
431      Si se encuentra al principio, entonces '*p' se posiciona sobre el puntero '*principio'; entonces, como
el primer
432      elemento se va a eliminar, el número siguiente pasará a ser el primero; por lo tanto principio = p ->
siguiente.
433      */
434      if(donde == PRINCIPIO){
435          p = principio;
436          principio = p -> siguiente;
437      }
438      /*
439      Si se encuentra en medio, entonces utilizamos al puntero '*anterior' para poner a '*p' sobre el
elemento que queremos
440      eliminar; después, se "puentea" el apuntador '*siguiente' de la caja con dirección '*anterior'
441      */
442      else if(donde == ENMEDIO){
443          p = anterior -> siguiente;
444          anterior -> siguiente = p -> siguiente;
445      }
446      /*
447      Si no se encontró al principio ni enmedio, entonces se encontró al final. Utilizamos el puntero
'*anterior' para posicionarnos sobre
448      el último elemento; después, 'anterior -> siguiente' será NULL, porque después del elemento '*anterior'
ahora ya no habrá nada.
449      */
450      else{
451          p = anterior -> siguiente;
452          anterior -> siguiente = NULL;
453      }
```

```
454
455      delete(p); //Se elimina el dato deseado.
456      cuantos--; //Se decrementa el número de datos dentro de la lista.
457      return (1); //Se regresa un 1 porque sí pudimos eliminar el dato.
458  }
459
//-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
460
461  void lista_nodos::pintar(){
462      caja2 *p; p = principio; //Se crea un puntero tipo caja y se sitúa al principio de la lista ordenada.
463      while(p){ //Se imprimen todos los valores que forman parte de la estructura.
464         std::cout << "NODO: " << p -> numNodo << std::endl;
465          std::cout << "ENTRANTES: ";
466          (p -> entrantes).pintar();
467          std::cout << std::endl << "SALIENTES: ";
468          (p -> salientes).pintar();
469          std::cout << std::endl << "/*****************************************************************/" <<
std::endl;
470          p = p -> siguiente;
471      }
472  }
473
//-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
474
475  caja2* lista_nodos::lugar_agregado(){
476      return(lugarAgregado);
477  }
478
//-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
479
480  lista_nodos::~lista_nodos(){
481      caja2 *p;
482      while(principio){
483          p = principio;
484          principio = p -> siguiente;
485          (p -> entrantes).terminar();
486          (p -> salientes).terminar();
487          delete p;
488      }
489
490      principio = anterior = lugarAgregado = NULL;
491      encontrado = NO;
492      donde = VACIO;
493      cuantos = 0;
494      return;
495  }
496
//-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
497
498  void lista_nodos::terminar(){
499      caja2 *p;
500      while(principio){
501          p = principio;
502          principio = p -> siguiente;
503          (p -> entrantes).terminar();
504          (p -> salientes).terminar();
505          delete p;
506      }
507
508      principio = anterior = lugarAgregado = NULL;
509      encontrado = NO;
510      donde = VACIO;
```

```
511      cuantos = 0;
512      return;
513  }
514
515
/*****************************************************************************************************************
**************************************************************************/
516
517  struct caja3{
518      caja2 *direccionNodo;
519      float longitud;
520      caja3 *siguiente, *antes;
521  };
522
523
/*****************************************************************************************************************
**************************************************************************/
524
525  class lista{
526       private:
527          caja3 *anterior, *principio, *fin, *lugarAgregado;
528          int encontrado;
529          int donde;
530          enum encontrado{SI, NO};
531          enum donde{PRINCIPIO, ENMEDIO, FINAL};
532
533      public:
534          lista();
535          void iniciar();
536          ~lista();
537          void terminar();
538          void buscar(float a);
539          int agregar(caja2 *q, float a);
540          caja2 * sacar();
541          int borrar(float a);
542          void pintar();
543          void ajustar (caja3 *p, float a);
544          caja3 *lugar_agregado();
545  };
546
//-----------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------
547
548  lista::lista(){
549      anterior = principio = fin = lugarAgregado = NULL;
550      encontrado = NO;
551      donde = VACIO;
552  }
553
//-----------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------
554
555  void lista::iniciar(){
556      anterior = principio = fin = lugarAgregado = NULL;
557      encontrado = NO;
558      donde = VACIO;
559  }
560
561  void lista::buscar(float a){
562      caja3 *p; //Puntero tipo caja3 que recorrerá la estructura.
563
564      //Si la estructura está vacía entonces se indica que el dato no se encontró y que la lista está vacia.
565      if(principio == NULL){
566          encontrado = NO;
567          donde = VACIO;
568          lugarAgregado = NULL;
```

```
569          return;
570      }
571
572      //Si la lista no está vacía, entonces se comenzará a recorrer desde el principio hasta el final.
573      p = principio;
574      while(p){
575
576          //Si se encuentra el dato, entonces la variable encontrado toma el valor de SI.
577          if(p->longitud == a){
578              encontrado = SI;
579              lugarAgregado = p;
580              if(p == principio){ //Si se encuentra al principio, entonces donde = PRINCIPIO.
581                  donde = PRINCIPIO;
582              }else if(p -> siguiente == NULL){ //Si la siguiente dirección es NULL, entonces significa que
estamos en el final de la lista.
583                  donde = FINAL;
584               }
585               else{ //Si no estamos al principio ni al final de la lista ordenada, entonces encontramos el
número enmedio de la estructura.
586                  donde = ENMEDIO;
587              }
588              return;
589          }
590          /*
591          Como estamos ordenando números de menor a mayor, entonces si encontramos algún número menor a 'a',
simplemente seguimos recorriendo
592          la estructura.
593          */
594          else if(p->longitud < a){
595              encontrado = NO; //Se indica que no se ha encontrado aún.
596              anterior = p; //El puntero '*anterior' toma el valor de la dirección en donde nos encontramos.
597              p = p -> siguiente; //Finalmente p se pone sobre la siguiente caja de la lista ordenada.
598          }
599          /*
600          Como estamos ordenando números de menor a mayor, entonces, si encontramos algún número mayor a 'a',
significa que nuestro nuevo
601          valor debe agregarse una posición antes de este número mayor con el que nos encontramos.
602          */
603          else{
604              encontrado = NO; //Se indica que 'a' no ha sido encontrado.
605              if(principio == p){ //Si el primer elemento que checamos resultó ser mayor, entonces significa
que 'a' debe agregarse al principio.
606                  donde = PRINCIPIO;
607              }
608              else{ //Si no lo encontramos al principio, entonces forzosamente debe de agregarse en algún
lugar enmedio de la estructura.
609                  donde = ENMEDIO;
610              }
611              return;
612          }
613      }
614      /*
615      Si el ciclo se acaba y se recorre toda la lista sin encontrar ningún número mayor, entonces significa
que el número que queremos
616      encontrar es mayor a todos los que existen dentro de la lista ordenada; por lo tanto, no ha sido
encontrado y debe agregarse al final.
617      */
618      encontrado = NO;
619      donde = FINAL;
620      return;
621  }
622
623
624   int lista::agregar(caja2 *q, float a){
```

```
625      caja3 *p; //Puntero tipo caja3 donde guardaremos el valor a agregar.
626      buscar(a); //Se busca 'a'.
627
628      p = new caja3; //Se crea una nueva caja3.
629      p -> direccionNodo = q;//Se introduce el valor de 'a' a la caja.
630      p->longitud = a;
631
632      /*
633      Si la lista está vacía, entonces los dos punteros de la caja son NULL (porque no existe nada antes ni
después de éste elemento).
634      Este primer elemento también representará el principio y el fin de la lista, por eso principio = fin  =
p.
635      */
636
637      if(donde == VACIO){
638          p -> siguiente = NULL;
639          p -> antes = NULL;
640          principio = p;
641          fin = p;
642      }
643      /*
644      Si se debe agregar al principio, entonces '*siguiente' de la caja que agregaremos apuntará al elemento
que antes era el principio;
645      el '*anterior' de la cajita a agregar será NULL (No habrá elemnto antes que el primero de la lista). Se
conecta el puntero '*anterior'
646      del siguiente dato con la cajita que hemos agregado. El puntero '*principio' toma también el valor de
p.
647      */
648      else if(donde == PRINCIPIO){
649          p -> siguiente = principio;
650          p -> antes = NULL;
651          ( p -> siguiente ) -> antes = p;
652          principio = p;
653      }
654      /*
655      Si se debe agregar al final, entonces '*siguiente' de la caja que agregaremos debe ser NULL (no existe
ningún elemento después del último).
656      '*antes' de la caja que agregaremos deberá apuntar al que antes era el último elemento de la lista
(*fin). Se conecta el puntero '*siguiente'
657      de la caja que antes era la última con la nueva caja. El puntero '*fin' toma el valor de p.
658      */
659      else if(donde == FINAL){
660          p -> siguiente = NULL;
661          p -> antes = fin;
662          (p -> antes) -> siguiente = p;
663          fin = p;
664      }
665      /*
666      Si se agrega enmedio de la lista, entonces se puentean los punteros usando la dirección '*anterior'
667      */
668      else{
669          p -> siguiente = anterior -> siguiente;
670          p -> antes = anterior;
671          (p -> siguiente) -> antes = p;
672          anterior -> siguiente = p;
673      }
674      lugarAgregado = p;
675      return(1);
676  }
677
//-------------------------------------------------------------------------------------------------------------
-------------------------------------
678
679  caja2 *lista::sacar(){
680
681      if(principio == NULL && fin == NULL) return (NULL);
```

```
682      caja3 *p;
683      caja2 *q;
684
685      p = principio;
686      q = p->direccionNodo;
687      principio = p -> siguiente;
688      if (p -> siguiente == NULL) fin = NULL;
689      else (p -> siguiente) -> antes = principio;
690
691      delete(p);
692      return(q);
693  }
694
//-----------------------------------------------------------------------------------------------------------------------
-------------------------------------
695
696  caja3 *lista::lugar_agregado(){
697      return(lugarAgregado);
698  }
699
700
//-----------------------------------------------------------------------------------------------------------------------
-------------------------------------
701
702  int lista::borrar(float a){
703      caja3 *p;
704      buscar(a);
705      if(encontrado == NO) return(0);
706
707      if(donde == PRINCIPIO){
708          p = principio;
709          principio = p -> siguiente;
710          if (p -> siguiente == NULL) fin = NULL;
711          else (p -> siguiente) -> antes = principio;
712      }
713      else if(donde == ENMEDIO){
714          p = anterior -> siguiente;
715          anterior -> siguiente = p -> siguiente;
716          (p -> siguiente) -> antes = anterior;
717      }
718      else{
719          p = fin;
720          (p -> antes) -> siguiente = NULL;
721          fin = anterior;
722      }
723      delete(p);
724      return(1);
725  }
726
//-----------------------------------------------------------------------------------------------------------------------
-------------------------------------
727
728  void lista::ajustar(caja3 *p, float a){
729      if(p -> longitud <= a) return;
730
731      std::cout << "VOY A AJUSTAR EL NODO: " << p->direccionNodo->numNodo << "con la longitud " <<
p->longitud << std::endl;
732      caja2 *q; float ruta;
733      q = p->direccionNodo;
734
735      std::cout << "BORRE EL NODO: " << p->direccionNodo->numNodo << "con la longitud " << p->longitud <<
std::endl;
736      borrar(p->longitud);
737      pintar();
738      std::cout << "\n";
739      std::cout << "AGREGARE EL NODO: " << q->numNodo << "con la longitud " << a << std::endl;
```

```
740      agregar(q, a);
741      std::cout << "Lista que se supone que esta actualizada"<< std::endl;
742      pintar();
743      std::cout << "\n";
744
745      return;
746
747  }
748
//-------------------------------------------------------------------------------------------------------------------
-------------------------------------
749  void lista::pintar(){
750      caja3 *p;
751      p = principio;
752
753      while(p){
754           if(p == principio)std::cout << "Nodo: " << p->direccionNodo->numNodo << ",  Longitud: " <<
p->longitud << std::endl;
755           else std::cout << "Nodo: " << p->direccionNodo->numNodo << ", Longitud: " << p->longitud <<  ",
Antecesor: " << p->direccionNodo->antecesor->numNodo << std::endl;
756          p = p -> siguiente;
757      }
758      std::cout << "\b\b ";
759
760  }
761
762
//-------------------------------------------------------------------------------------------------------------------
-------------------------------------
763  lista::~lista(){
764      caja3 *p;
765      while(principio){
766          p = principio;
767          principio = p -> siguiente;
768          delete p;
769      }
770      anterior = principio = fin = lugarAgregado = NULL;
771      encontrado = NO;
772      donde = VACIO;
773
774      return;
775  }
776
//-------------------------------------------------------------------------------------------------------------------
-------------------------------------
777
778  void lista::terminar(){
779      caja3 *p;
780      while(principio){
781          p = principio;
782          principio = p -> siguiente;
783          delete p;
784      }
785      anterior = principio = fin = lugarAgregado = NULL;
786      encontrado = NO;
787      donde = VACIO;
788
789      return;
790  }
791
792
793
/*******************************************************************************************************************
*************************************************************************/
794  class grafica{
795      lista_nodos A;
```

```
796      lista L;
797      int existeRuta;
798  public:
799      grafica();
800      ~grafica();
801      void agregar_arco(int a, int b, float l);
802      void rutaCorta(int a, int b);
803      void pintar();
804  };
805
806  //-----------------------------------------------------------------------------------------------------------------------
807  grafica::grafica(){
808      A.iniciar();
809      L.iniciar();
810  }
811
812  //-----------------------------------------------------------------------------------------------------------------------
813  void grafica::agregar_arco(int a, int b, float l){
814      caja1 *p;
815      caja2 *q, *q2;
816      A.agregar(a);
817      q = q2 = A.lugar_agregado();
818      (q -> salientes).agregar(b);
819      p = (q->salientes).lugar_agregado();
820
821      A.agregar(b);
822      q = A.lugar_agregado();
823      (p -> direccionNodo) = q;
824      (p -> longitud) = l;
825      (q ->entrantes).agregar(a);
826      ((q->entrantes).lugar_agregado())->direccionNodo = q2;
827      ((q->entrantes).lugar_agregado())->longitud = l;
828  }
829
830  //-----------------------------------------------------------------------------------------------------------------------
831  void grafica::rutaCorta(int a, int b){
832      caja2 *p;
833      caja1 *q;
834
835      A.agregar(a);
836      p = A.lugar_agregado();
837
838      while(p){
839          p -> bandera = 2;
840          std::cout << "NODO DEFINITIVO: " << p->numNodo << std::endl << std::endl;
841
842          if(p->numNodo == b){
843              existeRuta = 1;
844              L.terminar();
845              while(p){
846                  L.agregar(p, p->rutaCorta);
847                  p = p->antecesor;
848              }
849              L.pintar();
850              return;
851          }
852
853          q = (p->salientes).Principio();
854          while(q){
855              std::cout << "Estoy en el nodo " << q->direccionNodo->numNodo << "con bandera: " <<
```

```
q->direccionNodo->bandera << std::endl;
856             std::cout << "LISTA: " << std::endl;
857             L.pintar();
858             std::cout << "\n";
859 
860             if( (q->direccionNodo)->bandera == 0 ){
861                 q->direccionNodo->bandera = 1;
862                 std::cout << "Nodo: " << q->direccionNodo->numNodo << ", " << "Bandera: " <<
q->direccionNodo->bandera << std::endl;
863                 (q->direccionNodo)->rutaCorta = (p->rutaCorta) + (q->longitud);
864                 std::cout << "Ruta corta: " << (p->rutaCorta) + (q->longitud) << std::endl;
865                 (q->direccionNodo)->antecesor = p;
866                 L.agregar( (q->direccionNodo), (p->rutaCorta) + (q->longitud) );
867                 std::cout << "LISTA: " << std::endl;
868                 L.pintar();
869                 std::cout << "\n";
870                 (q->direccionNodo)->direccion = L.lugar_agregado();
871             }
872             else if( (q->direccionNodo)->bandera == 1 && (p->rutaCorta)+(q->longitud) <
(q->direccionNodo)->rutaCorta ){
873                 L.ajustar( (q->direccionNodo)->direccion, (p->rutaCorta) + (q->longitud) );
874                 q->direccionNodo->antecesor = p;
875                 q->direccionNodo->rutaCorta = (p->rutaCorta) + (q->longitud);
876                 std::cout << "LISTA: " << std::endl;
877                 L.pintar();
878                 std::cout << "\n";
879 
880             }
881             q = q->siguiente;
882         }
883         p = L.sacar();
884     }
885     if(existeRuta == 1){
886         std::cout << "RUTA MAS CORTA: " << std::endl;
887         L.pintar();
888     }
889     else std::cout << "No existe ruta mas corta." << std::endl;
890 }
891 
892 
//-----------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------
893 
894 void grafica::pintar(){
895     A.pintar();
896 }
897 
//-----------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------
898 
899 grafica::~grafica(){
900     A.terminar();
901     L.terminar();
902 }
903 
904 #endif // GRAFICAS_4_H_INCLUDED
```