

```

1  /*
2  * File:   main.cpp
3  * Author: ANA RONCAL
4  * Created on 18 de septiembre de 2023, 05:39 PM
5  */
6
7  #include <iostream>
8  #include <cstdlib>
9  #include "ArbolBinario.h"
10 #include "ArbolBinarioBusqueda.h"
11 using namespace std;
12 #include "funcionesArbolesBB.h"
13 #include "funcionesArbolesBinarios.h"
14
15 /*
16  * IMPLEMENTA ARBOLES DE BÚSQUEDA BINARIA
17  */
18 int main(int argc, char** argv) {
19
20     struct ArbolBinarioBusqueda arbol, arbol2;
21     int auxiliarElemento;
22
23     construir(arbol);
24     insertar(arbol, 100);
25     insertar(arbol, 50);
26     insertar(arbol, 200);
27     insertar(arbol, 25);
28     insertar(arbol, 75);
29
30     preOrden(arbol);
31     enOrden(arbol);
32     postOrden(arbol);
33
34     construir(arbol2);
35     insertar(arbol2, 5);
36     insertar(arbol2, 15);
37     insertar(arbol2, 3);
38     insertar(arbol2, 7);
39     insertar(arbol2, 10);
40
41     preOrden(arbol2);
42     enOrden(arbol2);
43     postOrden(arbol2);
44
45     auxiliarElemento = minimoABB(arbol);
46     cout<<"Mínimo ABB: "<<auxiliarElemento<<endl;
47
48     auxiliarElemento = maximoABB(arbol);
49     cout<<"Máximo ABB: "<<auxiliarElemento<<endl;
50
51     cout<<"Sumar nodos: "<<sumarNodos(arbol)<<endl;
52
53     borraNodo(arbol, 100);
54     recorridoPorNivel(arbol);
55     enOrden(arbol);
56
57     destruirArbolBB(arbol);
58     destruirArbolBB(arbol2);
59
60
61     return 0;
62 }
63 /*
64 * File:   ArbolBB.h
65 * Author: ANA RONCAL
66 * Created on 18 de septiembre de 2023, 06:01 PM
67 */
68
69 #ifndef ARBOLBB_H

```

```

70  #define ARBOLBB_H
71
72  struct ArbolBinario{
73      struct NodoArbol * raiz;
74  };
75
76  #endif /* ARBOLBB_H */
77
78
79  /*
80  * File:    Nodo.h
81  * Author:  ANA RONCAL
82  * Created on 18 de septiembre de 2023, 05:55 PM
83  */
84
85  #ifndef NODO_H
86  #define NODO_H
87
88  struct NodoArbol{
89      int elemento; //Este dato representa el Elemento
90      struct NodoArbol * izquierda; //puntero al hijo izquierdo
91      struct NodoArbol * derecha; //puntero al hijo derecho
92  };
93
94  #endif /* NODO_H */
95
96  /*
97  * File:    Nodo.h
98  * Author:  ANA RONCAL
99  * Created on 12 de septiembre de 2023, 09:31 PM
100  */
101
102  #ifndef NODO_H
103  #define NODO_H
104
105  struct Nodo{
106      struct NodoArbol * nodo; /*aquí se cambia por el Elemento que se desee manejar*/
107      struct Nodo * siguiente;
108  };
109
110  #endif /* NODO_H */
111
112  /*
113  * File:    Lista.h
114  * Author:  ANA RONCAL
115  * Created on 12 de septiembre de 2023, 09:32 PM
116  */
117
118  #ifndef LISTA_H
119  #define LISTA_H
120
121  struct Lista{
122      struct Nodo * cabeza;
123      struct Nodo * cola;
124      int longitud;
125  };
126
127  #endif /* LISTA_H */
128
129  /*
130  * File:    Lista.h
131  * Author:  ANA RONCAL
132  * Created on 3 de septiembre de 2023, 01:28 AM
133  */
134
135  #ifndef LISTA_H
136  #define LISTA_H
137
138  struct Lista{

```

```

139     struct Nodo * cabeza;
140     int longitud;
141 };
142
143 #endif /* LISTA_H */
144
145 /*
146  * File:   Pilas.h
147  * Author: ANA RONCAL
148  * Created on 3 de septiembre de 2023, 01:33 AM
149  */
150
151 #ifndef PILAS_H
152 #define PILAS_H
153
154 struct Pila{
155     struct Lista lista;
156 };
157
158 #endif /* PILAS_H */
159
160 /*
161  * File:   Cola.h
162  * Author: ANA RONCAL
163  * Created on 12 de septiembre de 2023, 09:32 PM
164  */
165
166 #ifndef COLA_H
167 #define COLA_H
168
169 struct Cola{
170     struct Lista lista;
171 };
172
173 #endif /* COLA_H */
174
175
176 /*
177  * File:   funcionesLista.h
178  * Author: ANA RONCAL
179  * Created on 3 de septiembre de 2023, 01:32 AM
180  */
181
182 #ifndef FUNCIONESLISTA_H
183 #define FUNCIONESLISTA_H
184
185
186 void construirP(struct Lista &);
187 void insertarAlInicio(struct Lista &, struct NodoArbol *);
188 struct Nodo * crearNodoP(struct NodoArbol *, struct Nodo *);
189 struct NodoArbol * retornaCabezaP( struct Lista );
190 bool esListaVacíaP( struct Lista );
191 int longitudP( struct Lista );
192 void eliminaCabezaP(struct Lista &);
193
194 void destruir(struct Lista &);
195 void imprimeP( struct Lista );
196
197 #endif /* FUNCIONESLISTA_H */
198
199 /*
200  * File:   funcionesLista.h
201  * Author: ANA RONCAL
202  * Created on 12 de septiembre de 2023, 09:32 PM
203  */
204
205 #ifndef FUNCIONESLISTA_H
206 #define FUNCIONESLISTA_H
207

```

```

208 void construir(struct Lista &);
209 bool esListaVacia( struct Lista lista);
210 int longitud(struct Lista tad );
211 struct Nodo * crearNodo(struct NodoArbol *, struct Nodo * siguiente);
212 void insertarAlFinal(struct Lista & lista, struct NodoArbol *);
213 void imprime( struct Lista lista);
214 void eliminaCabeza(struct Lista & lista);
215 struct NodoArbol * retornaCabeza( struct Lista lista);
216 void destruirLista(struct Lista &);
217
218 #endif /* FUNCIONESLISTA_H */
219
220
221 /*
222  * File: funcionesCola.h
223  * Author: ANA RONCAL
224  * Created on 12 de septiembre de 2023, 09:32 PM
225  */
226
227 #ifndef FUNCIONESCOLA_H
228 #define FUNCIONESCOLA_H
229
230 void construir(struct Cola & cola);
231 bool esColaVacia( struct Cola cola);
232 void encolar(struct Cola & cola, struct NodoArbol *);
233 struct NodoArbol * desencolar(struct Cola & cola);
234 int longitud(struct Cola cola);
235 void imprime( struct Cola cola);
236 void simularFilaEspera(struct Cola & cola);
237 void destruirCola(struct Cola & cola);
238
239 #endif /* FUNCIONESCOLA_H */
240
241 /*
242  * File: funcionesPilas.h
243  * Author: ANA RONCAL
244  * Created on 3 de septiembre de 2023, 01:29 AM
245  */
246
247 #ifndef FUNCIONESPILAS_H
248 #define FUNCIONESPILAS_H
249
250 void construir(struct Pila & );
251 int longitud(struct Pila );
252 bool esPilaVacia( struct Pila );
253 void apilar(struct Pila &, struct NodoArbol * );
254 struct NodoArbol * desapilar(struct Pila &);
255 struct NodoArbol * cima( struct Pila );
256 void destruirPila(struct Pila );
257 void imprimir( struct Pila );
258
259 #endif /* FUNCIONESPILAS_H */
260
261
262
263
264 /*
265  * File: ArbolBinarioBusqueda.h
266  * Author: ANA RONCAL
267  * Created on 24 de septiembre de 2023, 10:33 PM
268  */
269
270 #ifndef ARBOLBINARIOBUSQUEDA_H
271 #define ARBOLBINARIOBUSQUEDA_H
272
273 struct ArbolBinarioBusqueda{
274     struct ArbolBinario arbolBinario;
275 };
276

```

```

277 #endif /* ARBOLBINARIOBUSQUEDA_H */
278
279 /*
280  * File:   funcionesArboles.h
281  * Author: ANA RONCAL
282  * Created on 18 de septiembre de 2023, 06:00 PM
283  */
284
285 #ifndef FUNCIONESARBOLES_H
286 #define FUNCIONESARBOLES_H
287
288 void construir(struct ArbolBinario & );
289
290 bool esArbolVacio(struct ArbolBinario arbol);
291 bool esNodoVacio(struct NodoArbol * nodo);
292
293 struct NodoArbol * crearNuevoNodoArbol(struct NodoArbol *, int,
294                                       struct NodoArbol *);
295 void plantarArbolBinario(struct ArbolBinario &, struct NodoArbol *, int,
296                          struct NodoArbol * );
297 void plantarArbolBinario(struct ArbolBinario &, struct ArbolBinario, int,
298                          struct ArbolBinario );
299 int raiz(struct NodoArbol * nodo);
300 void imprimeRaiz(struct ArbolBinario arbol);
301 void imprimeNodo(struct NodoArbol * nodo);
302
303 struct NodoArbol * hijoDerecho(struct ArbolBinario );
304 struct NodoArbol * hijoIzquierdo(struct ArbolBinario );
305
306 void recorrerEnPreOrdenRecursivo(struct NodoArbol * nodo);
307 void recorrerEnOrdenRecursivo(struct NodoArbol * nodo);
308 void recorrerEnPostOrdenRecursivo(struct NodoArbol * nodo);
309
310 void recorrerEnOrden(struct ArbolBinario );
311 void recorrerPreOrden(struct ArbolBinario );
312 void recorrerPostOrden(struct ArbolBinario );
313
314 int altura(struct ArbolBinario);
315 int alturaRecursivo(struct NodoArbol * nodo);
316
317 int numeroNodos(struct ArbolBinario );
318 int numeroNodosRecursivo(struct NodoArbol * nodo);
319
320 int numeroHojas(struct ArbolBinario );
321
322 int sumarNodosRecursivo(struct NodoArbol * nodo);
323 int sumarNodos(struct ArbolBinario arbol);
324
325 int esEquilibrado(struct ArbolBinario );
326 int esEquilibradoRecursivo(struct NodoArbol * nodo);
327
328 int esHoja(struct ArbolBinario );
329
330 void destruirArbolBinario(struct ArbolBinario );
331 void destruirRecursivo(struct NodoArbol *);
332
333 void recorridoPorNivel(struct ArbolBinario arbol);
334 void enOrdenIterativo(struct ArbolBinario arbol);
335 void preOrdenIterativo(struct ArbolBinario arbol);
336
337
338 #endif /* FUNCIONESARBOLES_H */
339
340 /*
341  * File:   funcionesArbolesBB.h
342  * Author: ANA RONCAL
343  * Created on 24 de septiembre de 2023, 10:34 PM
344  */
345

```

```

346 #ifndef FUNCIONESARBOLESBB_H
347 #define FUNCIONESARBOLESBB_H
348
349 void construir(struct ArbolBinarioBusqueda & arbol);
350 void insertar(struct ArbolBinarioBusqueda & arbol, int elemento);
351 bool esArbolVacio(struct ArbolBinarioBusqueda arbol);
352 void insertarRecursivo(struct NodoArbol *& raiz, int elemento);
353 void plantarArbolBB(struct NodoArbol *& arbol,
354                     struct NodoArbol * arbolIzquierdo, int elemento,
355                     struct NodoArbol * arbolDerecho);
356 void preOrden(struct ArbolBinarioBusqueda arbol);
357 void enOrden(struct ArbolBinarioBusqueda arbol);
358 void postOrden(struct ArbolBinarioBusqueda arbol);
359
360 void destruirArbolBB(struct ArbolBinarioBusqueda arbol);
361
362 int minimoABB(struct ArbolBinarioBusqueda );
363 int minimoABBRecursivo(struct NodoArbol * nodo);
364 int maximoABBRecursivo(struct NodoArbol * nodo);
365 int maximoABB(struct ArbolBinarioBusqueda );
366 struct NodoArbol * minimoArbol(struct NodoArbol * nodo);
367
368 bool buscaArbolRecursivo(struct NodoArbol * nodo, int dato);
369 bool buscaArbol(struct ArbolBinarioBusqueda arbol ,int dato);
370
371 void borraNodo(struct ArbolBinarioBusqueda,int );
372 struct NodoArbol * borraNodoRecursivo(struct NodoArbol * nodo, int elemento);
373 int comparaABB(int , int );
374
375 int sumarNodos(struct ArbolBinarioBusqueda arbol);
376
377 void recorridoPorNivel(struct ArbolBinarioBusqueda );
378
379 #endif /* FUNCIONESARBOLESBB_H */
380
381
382
383 /*
384  * File:   funcionesLista.cpp
385  * Author: ANA RONCAL
386  * Created on 12 de septiembre de 2023, 09:32 PM
387  */
388
389 #include <iostream>
390 #include "ArbolBinario.h"
391 #include "Nodo.h"
392 #include "NodoArbol.h"
393 #include "Lista.h"
394 using namespace std;
395 #include "funcionesLista.h"
396 #include "funcionesArbolesBinarios.h"
397 #include "Nodo.h"
398 #include "Lista.h"
399
400
401 /*Valores iniciales de la lista*/
402 void construir(struct Lista & lista){
403     lista.cabeza = nullptr;
404     lista cola = nullptr;
405     lista.longitud = 0;
406 }
407
408 /*devuelve si la lista esta vacia 1, caso contrario 0 */
409 bool esListaVacia( struct Lista lista){
410     return lista.cabeza == nullptr;
411 }
412
413 /*DEVUELVE LA CANTIDAD DE ELEMENTOS DE LA LISTA*/
414 int longitud(struct Lista tad ){

```

```

415     return tad.longitud;
416 }
417
418 /*CREA UN NUEVO ELEMENTO CON VALORES INICIALES*/
419 struct Nodo * crearNodo(struct NodoArbol * nodo, struct Nodo * siguiente){
420
421     struct Nodo * nuevoNodo = new struct Nodo;
422     nuevoNodo->nodo = nodo;
423     nuevoNodo->siguiente = siguiente;
424     return nuevoNodo;
425 }
426
427 /*INSERTA UN ELEMENTO AL FINAL DE LA LISTA*/
428 void insertarAlFinal(struct Lista & lista, struct NodoArbol * nodo){
429
430     struct Nodo * nuevoNodo = crearNodo(nodo, nullptr);
431     Nodo * ultimoNodo = lista cola; /*obtiene el último nodo*/
432     if (ultimoNodo == nullptr){
433         lista.cabeza = nuevoNodo;
434         lista.cola = nuevoNodo;
435     }
436     else{
437         ultimoNodo->siguiente = nuevoNodo;
438         lista.cola = nuevoNodo;
439     }
440     lista.longitud++;
441 }
442
443 struct NodoArbol * retornaCabeza( struct Lista lista){
444     if (esListaVacia(lista)){
445         cout<<"No existe la cabeza por que la cola está vacía"<<endl;
446         exit(1);
447     }
448
449     return lista.cabeza->nodo;
450 }
451
452 /*ELIMINA EL PRIMER ELEMENTO DE LA LISTA*/
453 void eliminaCabeza(struct Lista & lista){
454     struct Nodo * nodoEliminar = lista.cabeza;
455     if (nodoEliminar == nullptr ){
456         cout<<"No se puede eliminar la cabeza pues la lista está vacía";
457         exit(1);
458     }
459     else{
460         lista.cabeza = lista.cabeza->siguiente;
461         if(lista.cabeza == nullptr)
462             lista.cola = nullptr;
463         delete nodoEliminar;
464         lista.longitud--;
465     }
466 }
467
468 /*LIBERA LA MEMORIA*/
469 void destruirLista(struct Lista & tad){
470     struct Nodo * recorrido = tad.cabeza;
471     struct Nodo * eliminarNodo;
472
473     while(recorrido != nullptr){
474         eliminarNodo = recorrido;
475         recorrido = recorrido->siguiente;
476         delete eliminarNodo;
477     }
478     tad.cabeza = nullptr;
479     tad.cola = nullptr;
480     tad.longitud = 0;
481 }
482
483 void imprime(struct Lista lista){

```

```

484
485     if (esListaVacia(lista)){
486         cout<<"La cola esta vacía"<<endl;
487     }
488     else{
489         struct Nodo * recorrido = lista.cabeza;
490         while(recorrido != nullptr){
491             imprimeNodo(recorrido->nodo);
492             recorrido = recorrido->siguiente;
493         }
494     }
495     cout<<endl;
496 }
497
498 /*
499  * File:   funcionesLista.cpp
500  * Author: ANA RONCAL
501  * Created on 3 de septiembre de 2023, 01:32 AM
502  */
503
504 #include <iostream>
505 #include "ArbolBinario.h"
506 #include "Nodo.h"
507 #include "NodoArbol.h"
508 #include "ListaP.h"
509 using namespace std;
510 #include "funcionesListaP.h"
511 #include "funcionesArbolesBinarios.h"
512 #include "Nodo.h"
513 #include "ListaP.h"
514
515 /*Valores iniciales de la lista*/
516 void construirP(struct Lista & tad){
517     tad.cabeza = nullptr;
518     tad.longitud = 0;
519 }
520
521 /*devuelve si la lista esta vacia 1, caso contrario 0 */
522 bool esListaVaciaP(struct Lista tad){
523     return tad.cabeza == nullptr;
524 }
525
526 /*devuelve la longitud de la lista*/
527 int longitudP( struct Lista tad){
528     return tad.longitud;
529 }
530
531 struct NodoArbol * retornaCabezaP(struct Lista tad){
532     if (esListaVaciaP(tad)){
533         cout<<"No existe la cabeza por que la cola está vacía"<<endl;
534         exit(1);
535     }
536     return tad.cabeza->nodo;
537 }
538
539 /*inserta un nodo siempre al inicio de la lista*/
540 void insertarAlInicio(struct Lista & tad, struct NodoArbol * nodo){
541
542     /*Crea un nuevo nodo*/
543     struct Nodo * nuevoNodo = new struct Nodo;
544     nuevoNodo = crearNodoP(nodo, tad.cabeza);
545
546     tad.cabeza = nuevoNodo;
547     tad.longitud++;
548 }
549
550 /*Crea un nuevo nodo con los datos dados como parÃ;metros*/
551 struct Nodo * crearNodoP(struct NodoArbol * nodo, struct Nodo * siguiente){
552

```



```

553     struct Nodo * nuevoNodo = new struct Nodo;
554     nuevoNodo->nodo = nodo;
555     nuevoNodo->siguiente = siguiente;
556     return nuevoNodo;
557 }
558
559
560 void eliminaCabezaP(struct Lista & lista){
561     struct Nodo * nodoEliminar = lista.cabeza;
562     if (nodoEliminar == nullptr ){
563         cout<<"No se puede eliminar la cabeza pues la lista estÃ¡ vacÃ­a";
564         exit(1);
565     }
566     else{
567         lista.cabeza = lista.cabeza->siguiente;
568         delete nodoEliminar;
569         lista.longitud--;
570     }
571 }
572
573 void destruir(struct Lista & tad){
574     /*recorrido apunta al inicio del tad*/
575     struct Nodo * recorrido = tad.cabeza;
576
577     while(recorrido != nullptr){
578         /*Nodo auxiliar que va servir para eliminar los nodos*/
579         struct Nodo * nodoAEliminar = recorrido;
580         recorrido = recorrido->siguiente;
581         delete nodoAEliminar;
582     }
583     /*la lista queda vacia*/
584     tad.cabeza = nullptr;
585     tad.longitud = 0;
586 }
587
588 /*Recordar que las Pilas no se recorren en forma secuencial*/
589 /*Se va utilizar solo para mostrar los valores*/
590 void imprimeP( struct Lista tad){
591
592     if (esListaVaciaP(tad)){
593         cout<<"La Pila estÃ¡ vacÃ­a"<<endl;
594     }
595     else{
596
597         struct Nodo * recorrido = tad.cabeza;
598         int estaImprimiendoLaCabeza = 1;
599         cout<<"[";
600
601         while(recorrido != nullptr){
602             /*Este artificio coloca la primera coma despuÃ©s de la cabeza*/
603             if (!estaImprimiendoLaCabeza)
604                 cout<<" ";
605             estaImprimiendoLaCabeza = 0;
606             imprimeNodo(recorrido->nodo);
607             recorrido = recorrido->siguiente;
608         }
609         cout<<"]"<<endl;
610     }
611 }
612
613 /*
614  * File:   funcionesPilas.cpp
615  * Author: ANA RONCAL
616  * Created on 3 de septiembre de 2023, 01:29 AM
617  */
618
619 #include <iostream>
620 #include "ArbolBinario.h"
621 #include "Nodo.h"

```

```

622 #include "ListaP.h"
623 #include "Pila.h"
624 using namespace std;
625 #include "funcionesListaP.h"
626 #include "funcionesPila.h"
627
628 /*constructor de Pila*/
629 void construir(struct Pila & pila){
630     construirP(pila.lista);
631 }
632
633 /*Determina si la pila está vacía*/
634 bool esPilaVacía( struct Pila pila){
635     return esListaVacíaP(pila.lista);
636 }
637
638 /*Determina el número de elementos de la pila*/
639 int longitud( struct Pila pila){
640     return longitudP(pila.lista);
641 }
642
643 /*push, añade un elemento a la parte superior de la pila*/
644 void apilar(struct Pila & pila, struct NodoArbol * nodo){
645     insertarAlInicio(pila.lista, nodo);
646 }
647
648 /*pop, elimina un elemento de la parte superior de la pila*/
649 struct NodoArbol * desapilar(struct Pila & pila){
650     if (esPilaVacía(pila)){
651         cout<<"La pila está vacía, por lo tanto no se puede desapilar"<<endl;
652         exit(11);
653     }
654     struct NodoArbol * nodo = cima(pila);
655     eliminaCabezaP(pila.lista);
656     return nodo;
657 }
658
659 /*examina un elemento situado en la parte superior de la pila*/
660 struct NodoArbol * cima(struct Pila pila){
661     if (esPilaVacía(pila)){
662         cout<<"La pila está vacía por lo tanto no posee cima"<<endl;
663         exit(12);
664     }
665     return retornaCabezaP(pila.lista);
666 }
667
668
669 /*destruye la pila*/
670 void destruirPila(struct Pila pila){
671     destruir(pila.lista);
672 }
673
674 /*Recordar que las Pilas no se recorren en forma secuencial*/
675 /*Se va utilizar solo para mostrar los valores*/
676 void imprimir(const struct Pila & pila){
677     imprimeP(pila.lista);
678 }
679
680 ///*imprime desapilando*/
681 //void imprime(struct Pila & pila){
682 //
683 //     while(not esPilaVacía(pila)){
684 //         cout<<cima(pila)<<"-";
685 //         desapilar(pila);
686 //     }
687 //}
688
689 /*
690 * File:   funcionesCola.cpp

```

```

691  * Author: ANA RONCAL
692  * Created on 12 de septiembre de 2023, 09:32 PM
693  */
694
695  #include <iostream>
696  #include <iomanip>
697  #include "ArbolBinario.h"
698  #include "Nodo.h"
699  #include "Lista.h"
700  #include "Cola.h"
701  using namespace std;
702  #include "funcionesLista.h"
703  #include "funcionesCola.h"
704
705
706  #define PROCESO 120
707  #define MAX_CAJEROS 10
708  #define NUM_CLIENTES 100
709
710  void construir(struct Cola & cola){
711      construir(cola.lista);
712  }
713
714  bool esColaVacia( struct Cola cola){
715      return esListaVacia(cola.lista);
716  }
717
718  int longitud(struct Cola cola){
719      return longitud(cola.lista);
720  }
721
722  void encolar(struct Cola & cola, struct NodoArbol * nodo){
723      insertarAlFinal(cola.lista, nodo);
724  }
725
726  struct NodoArbol * desencolar(struct Cola & cola){
727      if(esColaVacia(cola)){
728          cout<<"La cola está vacía no se puede desencolar"<<endl;
729          exit(1);
730      }
731
732      struct NodoArbol * nodo = retornaCabeza(cola.lista);
733      eliminaCabeza(cola.lista);
734      return nodo;
735  }
736
737  void destruirCola(struct Cola & cola){
738      destruirLista(cola.lista);
739  }
740
741  void imprime( struct Cola cola){
742      imprime(cola.lista);
743  }
744
745  /*
746  * File: funcionesArbolesBB.cpp
747  * Author: ANA RONCAL
748  * Created on 19 de septiembre de 2023, 10:46 AM
749  */
750
751  #include <iostream>
752  #include <iomanip>
753  #include <fstream>
754  #include <cstring>
755  #include "NodoArbol.h"
756  #include "Nodo.h"
757  #include "ArbolBinario.h"
758  #include "Lista.h"
759  #include "ListaP.h"

```

```

760 #include "Cola.h"
761 #include "Pila.h"
762 using namespace std;
763 #include "funcionesArbolesBinarios.h"
764 #include "funcionesCola.h"
765 #include "funcionesPila.h"
766
767 void construir(struct ArbolBinario & arbol ){
768     arbol.raiz = nullptr;
769 }
770
771 bool esNodoVacio(struct NodoArbol * nodo){
772     return nodo == nullptr;
773 }
774
775 bool esArbolVacio( struct ArbolBinario arbol){
776     return esNodoVacio(arbol.raiz);
777 }
778
779 struct NodoArbol * crearNuevoNodoArbol(struct NodoArbol * arbolIzquierdo,
780                                     int elemento, struct NodoArbol * arbolDerecho){
781     struct NodoArbol * nuevoNodo = new struct NodoArbol;
782     nuevoNodo->elemento = elemento;
783     nuevoNodo->izquierda = arbolIzquierdo;
784     nuevoNodo->derecha = arbolDerecho;
785     return nuevoNodo;
786 }
787
788 void plantarArbolBinario(struct ArbolBinario & arbol, struct NodoArbol * arbolIzquierdo,
789                         int elemento, struct NodoArbol * arbolDerecho){
790
791     struct NodoArbol * nuevoNodo = crearNuevoNodoArbol(arbolIzquierdo, elemento,
792                                                         arbolDerecho);
793     arbol.raiz = nuevoNodo;
794 }
795
796 void plantarArbolBinario(struct ArbolBinario & arbol, struct ArbolBinario arbolIzquierdo,
797                         int elemento, struct ArbolBinario arbolDerecho){
798
799     struct NodoArbol * nuevoNodo = crearNuevoNodoArbol(arbolIzquierdo.raiz, elemento,
800                                                         arbolDerecho.raiz);
801     arbol.raiz = nuevoNodo;
802 }
803
804 int raiz(struct NodoArbol * nodo){
805     if (esNodoVacio(nodo)){
806         cout<<"No se puede obtener raíz de un árbol vacío"<<endl;
807         exit(1);
808     }
809     return nodo->elemento;
810 }
811
812 struct NodoArbol * hijoDerecho(struct ArbolBinario arbol){
813     if (esArbolVacio(arbol)){
814         cout<<"No se puede obtener raíz de un árbol vacío"<<endl;
815         exit(1);
816     }
817     return arbol.raiz->derecha;
818 }
819
820 struct NodoArbol * hijoIzquierdo(struct ArbolBinario arbol){
821     if (esArbolVacio(arbol)){
822         cout<<"No se puede obtener raíz de un árbol vacío"<<endl;
823         exit(1);
824     }
825     return arbol.raiz->izquierda;
826 }

```

```

827 void imprimeRaiz(struct ArbolBinario arbol){
828     imprimeNodo(arbol.raiz);
829 }
830
831 void imprimeNodo(struct NodoArbol * nodo){
832     cout<<setw(5)<<nodo->elemento;
833 }
834
835 void recorrerEnOrdenRecursivo(struct NodoArbol * nodo){
836     if(not esNodoVacio(nodo)){
837         recorrerEnOrdenRecursivo(nodo->izquierda);
838         imprimeNodo(nodo);
839         recorrerEnOrdenRecursivo(nodo->derecha);
840     }
841 }
842
843 /*En árbol, se lleva a cabo visitando el hijo izquierdo del nodo, luego el nodo
844 luego todos los restantes, comenzando por la raíz*/
845 void recorrerEnOrden(struct ArbolBinario arbol){
846     /*Imprime en orden*/
847     if (not esArbolVacio(arbol)){
848         recorrerEnOrdenRecursivo(arbol.raiz);
849     }
850     cout<<endl;
851 }
852
853 void recorrerEnPreOrdenRecursivo(struct NodoArbol * nodo){
854     if(not esNodoVacio(nodo)){
855         imprimeNodo(nodo);
856         recorrerEnPreOrdenRecursivo(nodo->izquierda);
857         recorrerEnPreOrdenRecursivo(nodo->derecha);
858     }
859 }
860
861 /*recorrido descendente, se lleva a cabo visitando cada nodo, seguido de sus hijos,
862 luego todos los restantes, comenzando por la raíz*/
863 void recorrerPreOrden(struct ArbolBinario arbol){
864     if (not esArbolVacio(arbol)){
865         recorrerEnPreOrdenRecursivo(arbol.raiz);
866     }
867     cout<<endl;
868 }
869
870 void recorrerEnPostOrdenRecursivo(struct NodoArbol * nodo){
871     if(not esNodoVacio(nodo)){
872         recorrerEnPostOrdenRecursivo(nodo->izquierda);
873         recorrerEnPostOrdenRecursivo(nodo->derecha);
874         imprimeNodo(nodo);
875     }
876 }
877
878 /*recorrido ascendente, se lleva a cabo visitando los hijos, y luego el nodo
879 luego todos los restantes, comenzando por la raíz*/
880 void recorrerPostOrden(struct ArbolBinario arbol){
881
882     if (not esArbolVacio(arbol)){
883         recorrerEnPostOrdenRecursivo(arbol.raiz);
884     }
885     cout<<endl;
886 }
887
888 int maximo(int a, int b){
889     return a>=b ? a: b;
890 }
891
892 int alturaRecursivo(struct NodoArbol * nodo){
893     if(esNodoVacio(nodo))
894         return 0;
895     else if(esNodoVacio(nodo->izquierda) and esNodoVacio(nodo->derecha))

```

```

896         return 0;
897     else
898         return 1 + maximo( alturaRecursivo(nodo->izquierda), alturaRecursivo(nodo->
derecha));
899 }
900
901 int altura(struct ArbolBinario arbol){
902     return alturaRecursivo(arbol.raiz); //como el arbol ha sido construido no va apuntar
a nullptr
903 }
904
905 int numeroNodosRecursivo(struct NodoArbol * nodo){
906     if(esNodoVacio(nodo))
907         return 0;
908     else
909         return 1 + numeroNodosRecursivo(nodo->izquierda) + numeroNodosRecursivo(nodo->
derecha);
910 }
911
912 /*Determina el número de elementos del árbol*/
913 int numeroNodos(struct ArbolBinario arbol){
914     return numeroNodosRecursivo(arbol.raiz);
915 }
916
917 int numeroHojasRecursivo(struct NodoArbol * nodo){
918     if(esNodoVacio(nodo))
919         return 0;
920     else if ( esNodoVacio(nodo->izquierda) and esNodoVacio(nodo->derecha) )
921         return 1;
922     else
923         return numeroHojasRecursivo(nodo->izquierda) + numeroHojasRecursivo(nodo->derecha
);
924 }
925
926 int numeroHojas(struct ArbolBinario arbol){
927     return numeroHojasRecursivo(arbol.raiz);
928 }
929
930 int sumarNodosRecursivo(struct NodoArbol * nodo){
931     if(esNodoVacio(nodo))
932         return 0;
933     else
934         return nodo->elemento + sumarNodosRecursivo(nodo->izquierda) +
sumarNodosRecursivo(nodo->derecha);
935 }
936
937 int sumarNodos(struct ArbolBinario arbol){
938     return sumarNodosRecursivo(arbol.raiz);
939 }
940
941 int esEquilibradoRecursivo(struct NodoArbol * nodo){
942     if(esNodoVacio(nodo))
943         return 1;
944     else{
945         int alturaHijoIzquierdo = alturaRecursivo(nodo->izquierda);
946         int alturaHijoDerecho = alturaRecursivo(nodo->derecha);
947         int diferencia = abs(alturaHijoIzquierdo - alturaHijoDerecho);
948         return diferencia<=1 and
949             esEquilibradoRecursivo(nodo->izquierda) and
950             esEquilibradoRecursivo(nodo->derecha);
951     }
952 }
953
954 int esEquilibrado(struct ArbolBinario arbol ){
955     return esEquilibradoRecursivo(arbol.raiz);
956 }
957
958 int esHoja(struct ArbolBinario arbol){
959     if(esArbolVacio(arbol))

```

```

960         return 0;
961     else
962         return esNodoVacio(arbol.raiz->izquierda) and esNodoVacio(arbol.raiz->derecha);
963 }
964
965 void destruirArbolBinario(struct ArbolBinario arbol){
966     destruirRecursivo(arbol.raiz);
967     arbol.raiz = nullptr;
968 }
969
970 void destruirRecursivo(struct NodoArbol * nodo){
971     if(not (esNodoVacio(nodo))){
972         destruirRecursivo(nodo->izquierda);
973         destruirRecursivo(nodo->derecha);
974         delete nodo;
975         nodo = nullptr;
976     }
977 }
978
979 /*recorre el árbol por niveles usando una cola*/
980 void recorridoPorNivel(struct ArbolBinario arbol){
981     struct Cola cola; /*Se usa una cola para acceder a los nodos*/
982     construir(cola);
983     if(not esArbolVacio(arbol)){
984         encolar(cola, arbol.raiz);
985         while(not esColaVacia(cola)){
986             struct NodoArbol * nodo = desencolar(cola);
987             imprimeNodo(nodo);
988             if (not esNodoVacio(nodo->izquierda)){
989                 encolar(cola, nodo->izquierda);
990             }
991             if (not esNodoVacio(nodo->derecha)){
992                 encolar(cola, nodo->derecha);
993             }
994         }
995     }
996     cout<<endl;
997     destruirCola(cola);
998 }
999
1000 void enOrdenIterativo(struct ArbolBinario arbol){
1001     struct Pila pila; /*Se usa una pila para acceder a los nodos*/
1002     construir(pila);
1003     int fin = 0;
1004     do{
1005         while (not esArbolVacio(arbol)){
1006             apilar(pila, arbol.raiz);
1007             arbol.raiz = arbol.raiz->izquierda;
1008         }
1009         if (esPilaVacia(pila))
1010             fin = 1;
1011         else{
1012             arbol.raiz = desapilar(pila);
1013             imprimeRaiz(arbol);
1014             arbol.raiz = arbol.raiz->derecha;
1015         }
1016     } while(fin == 0);
1017     destruirPila(pila);
1018     cout<<endl;
1019 }
1020
1021 void preOrdenIterativo(struct ArbolBinario arbol){
1022     struct Pila pila; /*Se usa una pila para acceder a los nodos*/
1023     construir(pila);
1024     if (not esArbolVacio(arbol)){
1025         apilar(pila, arbol.raiz);
1026         while(not esPilaVacia(pila)){
1027             struct NodoArbol * nodo = desapilar(pila);
1028             imprimeNodo(nodo);

```

```

1029         if (not esNodoVacio(nodo->derecha))
1030             apilar(pila, nodo->derecha);
1031         if (not esNodoVacio(nodo->izquierda))
1032             apilar(pila, nodo->izquierda);
1033     }
1034 }
1035 cout<<endl;
1036 destruirPila(pila);
1037 }
1038
1039 /*
1040  * File:   funcionesArbolesBB.cpp
1041  * Author: ANA RONCAL
1042  * Created on 24 de septiembre de 2023, 10:36 PM
1043  */
1044
1045 #include <iostream>
1046 #include <iomanip>
1047 #include <fstream>
1048 #include <cstring>
1049 #include "ArbolBinario.h"
1050 #include "funcionesArbolesBB.h"
1051 #include "ArbolBinarioBusqueda.h"
1052 using namespace std;
1053 #include "funcionesArbolesBinarios.h"
1054 #include "NodoArbol.h"
1055
1056 void construir(struct ArbolBinarioBusqueda & arbol){
1057     construir(arbol.arbolBinario);
1058 }
1059
1060 bool esArbolVacio(struct ArbolBinarioBusqueda arbol){
1061     esArbolVacio(arbol.arbolBinario);
1062 }
1063
1064 void plantarArbolBB(struct NodoArbol *& arbol,
1065                    struct NodoArbol * arbolIzquierdo, int elemento,
1066                    struct NodoArbol * arbolDerecho){
1067
1068     struct NodoArbol * nuevoNodo = new struct NodoArbol;
1069     nuevoNodo->elemento = elemento;
1070     nuevoNodo->izquierda = arbolIzquierdo;
1071     nuevoNodo->derecha = arbolDerecho;
1072     arbol = nuevoNodo;
1073 }
1074 void insertar(struct ArbolBinarioBusqueda & arbol, int elemento){
1075     insertarRecurso(arbol.arbolBinario.raiz, elemento);
1076 }
1077
1078 void insertarRecurso(struct NodoArbol *& raiz, int elemento){
1079     if(raiz == nullptr)
1080         plantarArbolBB(raiz, nullptr, elemento, nullptr);
1081     else
1082         if (raiz->elemento > elemento)
1083             insertarRecurso(raiz->izquierda, elemento);
1084         else
1085             insertarRecurso(raiz->derecha, elemento);
1086 }
1087
1088 void enOrden(struct ArbolBinarioBusqueda arbol){
1089     recorrerEnOrden(arbol.arbolBinario);
1090 }
1091
1092 void preOrden(struct ArbolBinarioBusqueda arbol){
1093     recorrerPreOrden(arbol.arbolBinario);
1094 }
1095
1096 void postOrden(struct ArbolBinarioBusqueda arbol){
1097     recorrerPostOrden(arbol.arbolBinario);

```



```

1098 }
1099
1100 void destruirArbolBB(struct ArbolBinarioBusqueda arbol){
1101     destruirArbolBinario(arbol.arbolBinario);
1102 }
1103
1104 int minimoABBRecursoivo(struct NodoArbol * nodo){
1105     if(esNodoVacio(nodo)){
1106         cout<<"El árbol está vacío"<<endl;
1107         exit(1);
1108     }
1109     if(esNodoVacio(nodo->izquierda))
1110         return nodo->elemento;
1111     minimoABBRecursoivo(nodo->izquierda);
1112 }
1113
1114 int minimoABB(struct ArbolBinarioBusqueda arbol){
1115     return minimoABBRecursoivo(arbol.arbolBinario.raiz);
1116 }
1117
1118 int maximoABBRecursoivo(struct NodoArbol * nodo){
1119     if(esNodoVacio(nodo)){
1120         cout<<"El árbol está vacío"<<endl;
1121         exit(1);
1122     }
1123     if(esNodoVacio(nodo->derecha))
1124         return nodo->elemento;
1125     maximoABBRecursoivo(nodo->derecha);
1126 }
1127
1128 int maximoABB(struct ArbolBinarioBusqueda arbol){
1129     return maximoABBRecursoivo(arbol.arbolBinario.raiz);
1130 }
1131
1132 struct NodoArbol * minimoArbol(struct NodoArbol * nodo){
1133     if(esNodoVacio(nodo))
1134         return nodo;
1135     if(esNodoVacio(nodo->izquierda))
1136         return nodo;
1137     minimoArbol(nodo->izquierda);
1138 }
1139
1140 struct NodoArbol * borraNodoRecursoivo(struct NodoArbol * nodo, int dato){
1141     if(esNodoVacio(nodo))
1142         return nodo;
1143     if(comparaABB(nodo->elemento, dato) == 1){ //nodo mayor que el dato
1144         nodo->izquierda = borraNodoRecursoivo(nodo->izquierda, dato);
1145     }
1146     else{
1147         if(comparaABB(nodo->elemento, dato) == -1){ //nodo menor que el dato
1148             nodo->derecha = borraNodoRecursoivo(nodo->derecha, dato);
1149         }
1150         else{
1151             if(esNodoVacio(nodo->izquierda)){
1152                 struct NodoArbol * elimina = nodo->derecha;
1153                 delete elimina;
1154                 return elimina;
1155             }
1156             else{
1157                 if(esNodoVacio(nodo->derecha)){
1158                     struct NodoArbol * elimina = nodo->izquierda;
1159                     delete elimina;
1160                     return elimina;
1161                 }
1162                 else{
1163                     struct NodoArbol * temp = minimoArbol(nodo->derecha);
1164                     nodo->elemento = temp->elemento;
1165                     nodo->derecha = borraNodoRecursoivo(nodo->derecha, temp->elemento);
1166                 }
1167             }
1168         }
1169     }
1170 }

```

```
1167     }
1168 }
1169 return nodo;
1170 }
1171
1172 void borraNodo(struct ArbolBinarioBusqueda arbol, int dato){
1173     arbol.arbolBinario.raiz = borraNodoRecursoivo(arbol.arbolBinario.raiz, dato);
1174 }
1175
1176 int comparaABB(int elementoA, int elementoB ){
1177     if(elementoA == elementoB) return 0;
1178     else if(elementoA < elementoB) return -1;
1179     else if (elementoA > elementoB) return 1;
1180 }
1181
1182 bool buscaArbolRecursoivo(struct NodoArbol * nodo, int dato){
1183     if(esNodoVacio(nodo)){
1184         return false;
1185     }
1186     if(comparaABB(nodo->elemento, dato) == 0)
1187         return true;
1188     if(comparaABB(nodo->elemento, dato) == 1)
1189         return buscaArbolRecursoivo(nodo->izquierda, dato);
1190     else
1191         if(comparaABB(nodo->elemento, dato) == -1)
1192             return buscaArbolRecursoivo(nodo->derecha, dato);
1193 }
1194
1195 bool buscaArbol(struct ArbolBinarioBusqueda arbol ,int dato){
1196     return buscaArbolRecursoivo(arbol.arbolBinario.raiz, dato);
1197 }
1198 void recorridoPorNivel(struct ArbolBinarioBusqueda arbol){
1199     recorridoPorNivel(arbol.arbolBinario);
1200 }
1201
1202 int sumarNodos(struct ArbolBinarioBusqueda arbol){
1203     return sumarNodos(arbol.arbolBinario);
1204 }
1205
```