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

#include <iostream>
#include <cstdlib>
#include "ArbolBinario.h"
using namespace std;
#include "funcionesArbolesBinarios.h"
/*
 *
 */
int main(int argc, char** argv) {

    struct ArbolBinario arbol;
    construir(arbol);

    cout<<"Es árbol vacio: "<<esArbolVacio(arbol)<<endl;
    plantarArbolBinario(arbol, nullptr, 100, nullptr);
    imprimeRaiz(arbol); cout<<endl;
    struct ArbolBinario arbol1, arbol2, arbol3, arbol4;
    plantarArbolBinario(arbol1, nullptr, 25, nullptr);
    plantarArbolBinario(arbol2, nullptr, 75, nullptr);
    plantarArbolBinario(arbol3, arbol1, 50, arbol2);
    plantarArbolBinario(arbol4, nullptr, 150, nullptr);
    plantarArbolBinario(arbol, arbol3, 100, arbol4);
    cout<<"Es árbol vacio: "<<esArbolVacio(arbol)<<endl;

    cout<<"Recorrer en orden: "<<endl;
    recorrerEnOrden(arbol); cout<<endl;
    cout<<"Recorrer en pre orden: "<<endl;
    recorrerPreOrden(arbol); cout<<endl;
    cout<<"Recorrer en post orden: "<<endl;
    recorrerPostOrden(arbol); cout<<endl;

    cout<<"Recorrer por niveles: "<<endl;
    recorridoPorNivel(arbol);
    cout<<endl<<"Recorrer en orden iterativo: "<<endl;
    enOrdenIterativo(arbol);
    cout<<endl<<"Recorrer en Pre orden iterativo: "<<endl;
    preOrdenIterativo(arbol);

    cout<<endl;
    cout<<endl<<"ALTURA arbol: "<<altura(arbol)<<endl;
    cout<<"NÚMERO DE HOJAS: "<<numeroHojas(arbol)<<endl;
    cout<<"NÚMERO DE NODOS: "<<numeroNodos(arbol)<<endl;
    cout<<"ES EQUILIBRADO: "<<esEquilibrado(arbol)<<endl;

    destruirArbolBinario(arbol);
    cout<<"Es árbol vacio: "<<esArbolVacio(arbol)<<endl;

    return 0;
}

/*
 * File:   ArbolBB.h
 * Author: ANA RONCAL
 * Created on 18 de septiembre de 2023, 06:01 PM
 */

#ifndef ARBOLBB_H
#define ARBOLBB_H

struct ArbolBinario{
    struct NodoArbol * raiz;
};

```

```
#endif /* ARBOLBB_H */

/*
 * File:   Nodo.h
 * Author: ANA RONCAL
 * Created on 18 de septiembre de 2023, 05:55 PM
 */

#ifndef NODO_H
#define NODO_H

struct NodoArbol{
    int elemento;  //Este dato representa el Elemento
    struct NodoArbol * izquierda;  //puntero al hijo izquierdo
    struct NodoArbol * derecha;   //puntero al hijo derecho
};

#endif /* NODO_H */

/*
 * File:   Nodo.h
 * Author: ANA RONCAL
 * Created on 12 de septiembre de 2023, 09:31 PM
 */

#ifndef NODO_H
#define NODO_H

struct Nodo{
    struct NodoArbol * nodo; /*aquí se cambia por el Elemento que se desee manejar*/
    struct Nodo * siguiente;
};

#endif /* NODO_H */

/*
 * File:   Lista.h
 * Author: ANA RONCAL
 * Created on 3 de septiembre de 2023, 01:28 AM
 */

#ifndef LISTA_H
#define LISTA_H

struct Lista{
    struct Nodo * cabeza;
    int longitud;
};

#endif /* LISTA_H */

/*
 * File:   Lista.h
 * Author: ANA RONCAL
 * Created on 12 de septiembre de 2023, 09:32 PM
 */

#ifndef LISTA_H
#define LISTA_H

struct Lista{
    struct Nodo * cabeza;
    struct Nodo * cola;
    int longitud;
};

#endif /* LISTA_H */

/*
```

```
 * File:   Cola.h
 * Author: ANA RONCAL
 * Created on 12 de septiembre de 2023, 09:32 PM
 */

#ifndef COLA_H
#define COLA_H

struct Cola{
    struct Lista lista;
};

#endif /* COLA_H */

/*
 * File:   Pilas.h
 * Author: ANA RONCAL
 * Created on 3 de septiembre de 2023, 01:33 AM
 */

#ifndef PILAS_H
#define PILAS_H

struct Pila{
    struct Lista lista;
};

#endif /* PILAS_H */

/*
 * File:   funcionesLista.h
 * Author: ANA RONCAL
 * Created on 12 de septiembre de 2023, 09:32 PM
 */

#ifndef FUNCIONESLISTA_H
#define FUNCIONESLISTA_H

void construir(struct Lista &);
bool esListaVacia( struct Lista  lista);
int longitud(struct Lista tad );
struct Nodo * crearNodo(struct NodoArbol *, struct Nodo * siguiente);
void insertarAlFinal(struct Lista & lista, struct NodoArbol *);
void imprime( struct Lista  lista);
void eliminaCabeza(struct Lista  & lista);
struct NodoArbol * retornaCabeza( struct Lista  lista);
void destruirLista(struct Lista &);

#endif /* FUNCIONESLISTA_H */

/*
 * File:   funcionesLista.h
 * Author: ANA RONCAL
 * Created on 3 de septiembre de 2023, 01:32 AM
 */

#ifndef FUNCIONESLISTA_H
#define FUNCIONESLISTA_H


void construirP(struct Lista &);
void insertarAlInicio(struct Lista &, struct NodoArbol *);
struct Nodo * crearNodoP(struct NodoArbol *, struct Nodo *);
struct NodoArbol * retornaCabezaP( struct Lista );
bool esListaVaciaP( struct Lista );
int longitudP( struct Lista );
void eliminaCabezaP(struct Lista  &);

void destruir(struct Lista &);
```

```
208  void imprimeP( struct Lista );
209
210  #endif /* FUNCIONESLISTA_H */
211
212  /*
213   * File:   funcionesCola.h
214   * Author: ANA RONCAL
215   * Created on 12 de septiembre de 2023, 09:32 PM
216   */
217
218  #ifndef FUNCIONESCOLA_H
219  #define FUNCIONESCOLA_H
220
221  void construir(struct Cola & cola);
222  bool esColaVacia( struct Cola  cola);
223  void encolar(struct Cola & cola, struct NodoArbol *);
224  struct NodoArbol * desencolar(struct Cola & cola);
225  int longitud(struct Cola cola);
226  void imprime( struct Cola  cola);
227  void simularFilaEspera(struct Cola & cola);
228  void destruirCola(struct Cola & cola);
229
230  #endif /* FUNCIONESCOLA_H */
231
232  /*
233   * File:   funcionesPilas.h
234   * Author: ANA RONCAL
235   * Created on 3 de septiembre de 2023, 01:29 AM
236   */
237
238  #ifndef FUNCIONESPILAS_H
239  #define FUNCIONESPILAS_H
240
241  void construir(struct Pila & );
242  int longitud(struct Pila  );
243  bool esPilaVacia( struct Pila );
244  void apilar(struct Pila &, struct NodoArbol * );
245  struct NodoArbol * desapilar(struct Pila &);
246  struct NodoArbol * cima( struct Pila );
247  void destruirPila(struct Pila );
248  void imprimir( struct Pila );
249
250  #endif /* FUNCIONESPILAS_H */
251
252  /*
253   * File:   funcionesArboles.h
254   * Author: ANA RONCAL
255   * Created on 18 de septiembre de 2023, 06:00 PM
256   */
257
258  #ifndef FUNCIONESARBOLES_H
259  #define FUNCIONESARBOLES_H
260
261  void construir(struct ArbolBinario & );
262
263  bool esArbolVacio(struct ArbolBinario arbol);
264  bool esNodoVacio(struct NodoArbol * nodo);
265
266  struct NodoArbol * crearNuevoNodoArbol(struct NodoArbol *, int,
267                                                    struct NodoArbol *);
268  void plantarArbolBinario(struct ArbolBinario &, struct NodoArbol *, int,
269                                                   struct NodoArbol * );
270  void plantarArbolBinario(struct ArbolBinario &, struct ArbolBinario, int,
271                                                   struct ArbolBinario );
272  int raiz(struct NodoArbol * nodo);
273  void imprimeRaiz(struct ArbolBinario arbol);
274  void imprimeNodo(struct NodoArbol * nodo);
275
276  struct  NodoArbol * hijoDerecho(struct ArbolBinario );
```

```
277  struct  NodoArbol * hijoIzquierdo(struct ArbolBinario );
278
279  void recorrerEnPreOrdenRecursivo(struct NodoArbol * nodo);
280  void recorrerEnOrdenRecursivo(struct NodoArbol * nodo);
281  void recorrerEnPostOrdenRecursivo(struct NodoArbol * nodo);
282
283  void recorrerEnOrden(struct ArbolBinario );
284  void recorrerPreOrden(struct ArbolBinario );
285  void recorrerPostOrden(struct ArbolBinario );
286
287  int altura(struct ArbolBinario);
288  int alturaRecursivo(struct NodoArbol * nodo);
289
290  int numeroNodos(struct ArbolBinario );
291  int numeroNodosRecursivo(struct NodoArbol * nodo);
292
293  int numeroHojas(struct ArbolBinario );
294
295  int esEquilibrado(struct ArbolBinario );
296  int esEquilibradoRecursivo(struct NodoArbol * nodo);
297
298  int esHoja(struct ArbolBinario );
299
300  void destruirArbolBinario(struct ArbolBinario );
301  void destruirRecursivo(struct NodoArbol *);
302
303
304  void recorridoPorNivel(struct ArbolBinario arbol);
305  void enOrdenIterativo(struct ArbolBinario arbol);
306  void preOrdenIterativo(struct ArbolBinario arbol);
307
308
309  #endif /* FUNCIONESARBOLES_H */
310
311  /*
312   * File:   funcionesLista.cpp
313   * Author: ANA RONCAL
314   * Created on 12 de septiembre de 2023, 09:32 PM
315   */
316
317  #include <iostream>
318  #include "ArbolBinario.h"
319  #include "Nodo.h"
320  #include "NodoArbol.h"
321  #include "Lista.h"
322  using namespace std;
323  #include "funcionesLista.h"
324  #include "funcionesArbolesBinarios.h"
325  #include "Nodo.h"
326  #include "Lista.h"
327
328
329  /*Valores iniciales de la lista*/
330  void construir(struct Lista & lista){
331      lista.cabeza = nullptr;
332      lista.cola = nullptr;
333      lista.longitud = 0;
334  }
335
336  /*devuelve si la lista esta vacia 1, caso contrario 0 */
337  bool esListaVacia( struct Lista  lista){
338      return lista.cabeza == nullptr;
339  }
340
341  /*DEVUELVE LA CANTIDAD DE ELEMENTOS DE LA LISTA*/
342  int longitud(struct Lista tad ){
343      return tad.longitud;
344  }
345
```

```
346 /*CREA UN NUEVO ELEMENTO CON VALORES INICIALES*/
347 struct Nodo * crearNodo(struct NodoArbol * nodo, struct Nodo * siguiente){
348
349     struct Nodo * nuevoNodo = new struct Nodo;
350     nuevoNodo->nodo = nodo;
351     nuevoNodo->siguiente = siguiente;
352     return nuevoNodo;
353 }
354
355 /*INSERTA UN ELEMENTO AL FINAL DE LA LISTA*/
356 void insertarAlFinal(struct Lista & lista, struct NodoArbol * nodo){
357
358     struct Nodo * nuevoNodo = crearNodo(nodo, nullptr);
359     Nodo * ultimoNodo = lista.cola; /*obtiene el último nodo*/
360     if (ultimoNodo == nullptr){
361         lista.cabeza = nuevoNodo;
362         lista.cola = nuevoNodo;
363     }
364     else{
365         ultimoNodo->siguiente = nuevoNodo;
366         lista.cola = nuevoNodo;
367     }
368     lista.longitud++;
369 }
370
371 struct NodoArbol * retornaCabeza( struct Lista  lista){
372     if (esListaVacia(lista)){
373         cout<<"No existe la cabeza por que la cola está vacía"<<endl;
374         exit(1);
375     }
376
377     return lista.cabeza->nodo;
378 }
379
380 /*ELIMINA EL PRIMER ELEMENTO DE LA LISTA*/
381 void eliminaCabeza(struct Lista  & lista){
382     struct Nodo * nodoEliminar = lista.cabeza;
383     if (nodoEliminar == nullptr ){
384         cout<<"No se puede eliminar la cabeza pues la lista estÃ¡ vacÃ-a";
385         exit(1);
386     }
387     else{
388         lista.cabeza = lista.cabeza->siguiente;
389         if(lista.cabeza == nullptr)
390             lista.cola = nullptr;
391         delete nodoEliminar;
392         lista.longitud--;
393     }
394 }
395
396 /*LIBERA LA MEMORIA*/
397 void destruirLista(struct Lista & tad){
398     struct Nodo * recorrido = tad.cabeza;
399     struct Nodo * eliminarNodo;
400
401     while(recorrido != nullptr){
402         eliminarNodo = recorrido;
403         recorrido = recorrido->siguiente;
404         delete eliminarNodo;
405     }
406     tad.cabeza = nullptr;
407     tad.cola = nullptr;
408     tad.longitud = 0;
409 }
410
411 void imprime(struct Lista lista){
412
413     if (esListaVacia(lista)){
414         cout<<"La cola esta vacía"<<endl;
```

```
415         }
416         else{
417             struct Nodo * recorrido = lista.cabeza;
418             while(recorrido != nullptr){
419                 imprimeNodo(recorrido->nodo);
420                 recorrido = recorrido->siguiente;
421             }
422         }
423         cout<<endl;
424     }
425 
426     /*
427      * File:   funcionesLista.cpp
428      * Author: ANA RONCAL
429      * Created on 3 de septiembre de 2023, 01:32 AM
430      */
431 
432     #include <iostream>
433     #include "ArbolBinario.h"
434     #include "Nodo.h"
435     #include "NodoArbol.h"
436     #include "ListaP.h"
437     using namespace std;
438     #include "funcionesListaP.h"
439     #include "funcionesArbolesBinarios.h"
440     #include "Nodo.h"
441     #include "ListaP.h"
442 
443     /*Valores iniciales de la lista*/
444     void construirP(struct Lista & tad){
445         tad.cabeza = nullptr;
446         tad.longitud = 0;
447     }
448 
449     /*devuelve si la lista esta vacia 1, caso contrario 0 */
450     bool esListaVaciaP(struct Lista  tad){
451         return tad.cabeza == nullptr;
452     }
453 
454     /*devuelve la longitud de la lista*/
455     int longitudP( struct Lista tad){
456         return tad.longitud;
457     }
458 
459     struct NodoArbol * retornaCabezaP(struct Lista tad){
460         if (esListaVaciaP(tad)){
461             cout<<"No existe la cabeza por que la cola está vacía"<<endl;
462             exit(1);
463         }
464         return tad.cabeza->nodo;
465     }
466 
467     /*inserta un nodo siempre al inicio de la lista*/
468     void insertarAlInicio(struct Lista & tad, struct NodoArbol * nodo){
469 
470         /*Crea un nuevo nodo*/
471         struct Nodo * nuevoNodo = new struct Nodo;
472         nuevoNodo = crearNodoP(nodo, tad.cabeza);
473 
474         tad.cabeza = nuevoNodo;
475         tad.longitud++;
476     }
477 
478     /*Crea un nuevo nodo con los datos dados como parÃ¡metros*/
479     struct Nodo * crearNodoP(struct NodoArbol * nodo, struct Nodo * siguiente){
480 
481         struct Nodo * nuevoNodo = new struct Nodo;
482         nuevoNodo->nodo = nodo;
483         nuevoNodo->siguiente = siguiente;
```

```
484         return nuevoNodo;
485     }
486 
487 
488     void eliminaCabezaP(struct Lista  & lista){
489         struct Nodo * nodoEliminar = lista.cabeza;
490         if (nodoEliminar == nullptr ){
491             cout<<"No se puede eliminar la cabeza pues la lista estÃ¡ vacÃ-a";
492             exit(1);
493         }
494         else{
495             lista.cabeza = lista.cabeza->siguiente;
496             delete nodoEliminar;
497             lista.longitud--;
498         }
499     }
500 
501     void destruir(struct Lista & tad){
502         /*recorrido apunta al inicio del tad*/
503         struct Nodo * recorrido = tad.cabeza;
504 
505         while(recorrido != nullptr){
506             /*Nodo auxiliar que va servir para eliminar los nodos*/
507             struct Nodo * nodoAEliminar = recorrido;
508             recorrido = recorrido->siguiente;
509             delete nodoAEliminar;
510         }
511         /*la lista queda vacia*/
512         tad.cabeza = nullptr;
513         tad.longitud = 0;
514     }
515 
516     /*Recordar que las Pilas no se recorren en forma secuencial*/
517     /*Se va utilizar solo para mostrar los valores*/
518     void imprimeP( struct Lista  tad){
519 
520         if (esListaVaciaP(tad)){
521             cout<<"La Pila está vacía"<<endl;
522         }
523         else{
524 
525             struct Nodo * recorrido = tad.cabeza;
526             int estaImprimiendoLaCabeza = 1;
527             cout<<"[";
528 
529             while(recorrido != nullptr){
530                 /*Este artificio coloca la primera coma después de la cabeza*/
531                 if (!estaImprimiendoLaCabeza)
532                     cout<<", ";
533                 estaImprimiendoLaCabeza = 0;
534                 imprimeNodo(recorrido->nodo);
535                 recorrido = recorrido->siguiente;
536             }
537             cout<<"]"<<endl;
538         }
539     }
540 
541     /*
542      * File:   funcionesCola.cpp
543      * Author: ANA RONCAL
544      * Created on 12 de septiembre de 2023, 09:32 PM
545      */
546 
547     #include <iostream>
548     #include <iomanip>
549     #include "ArbolBinario.h"
550     #include "Nodo.h"
551     #include "Lista.h"
552     #include "Cola.h"
```

```
553  using namespace std;
554  #include "funcionesLista.h"
555  #include "funcionesCola.h"
556
557
558  #define PROCESO 120
559  #define MAX_CAJEROS 10
560  #define NUM_CLIENTES 100
561
562  void construir(struct Cola & cola){
563      construir(cola.lista);
564  }
565
566  bool esColaVacia( struct Cola  cola){
567      return esListaVacia(cola.lista);
568  }
569
570  int longitud(struct Cola cola){
571      return longitud(cola.lista);
572  }
573
574  void encolar(struct Cola & cola, struct NodoArbol * nodo){
575      insertarAlFinal(cola.lista, nodo);
576  }
577
578  struct NodoArbol *  desencolar(struct Cola & cola){
579      if(esColaVacia(cola)){
580          cout<<"La cola está vacía no se puede desencolar"<<endl;
581          exit(1);
582      }
583
584      struct NodoArbol * nodo = retornaCabeza(cola.lista);
585      eliminaCabeza(cola.lista);
586      return nodo;
587  }
588
589  void destruirCola(struct Cola & cola){
590      destruirLista(cola.lista);
591  }
592
593  void imprime( struct Cola  cola){
594      imprime(cola.lista);
595  }
596
597  /*
598   * File:   funcionesPilas.cpp
599   * Author: ANA RONCAL
600   * Created on 3 de septiembre de 2023, 01:29 AM
601   */
602
603  #include <iostream>
604  #include "ArbolBinario.h"
605  #include "Nodo.h"
606  #include "ListaP.h"
607  #include "Pila.h"
608  using namespace std;
609  #include "funcionesListaP.h"
610  #include "funcionesPila.h"
611
612  /*constructor de Pila*/
613  void construir(struct Pila & pila){
614      construirP(pila.lista);
615  }
616
617  /*Determina si la pila está vacía*/
618  bool esPilaVacia( struct Pila  pila){
619      return esListaVaciaP(pila.lista);
620  }
621
```

```
622  /*Determina el número de elementos de la pila*/
623  int longitud( struct Pila   pila){
624      return longitudP(pila.lista);
625  }
626
627  /*push, añade un elemento a la parte superior de la pila*/
628  void apilar(struct Pila & pila, struct NodoArbol * nodo){
629      insertarAlInicio(pila.lista, nodo);
630  }
631
632  /*pop, elimina un elemento de la parte superior de la pila*/
633  struct NodoArbol * desapilar(struct Pila & pila){
634      if (esPilaVacia(pila)){
635          cout<<"La pila está vacía, por lo tanto no se puede desapilar"<<endl;
636      exit(11);
637      }
638      struct NodoArbol * nodo = cima(pila);
639      eliminaCabezaP(pila.lista);
640      return nodo;
641  }
642
643  /*examina un elemento situado en la parte superior de la pila*/
644  struct NodoArbol * cima(struct Pila pila){
645      if (esPilaVacia(pila)){
646      cout<<"La pila está vacía por lo tanto no posee cima"<<endl;
647      exit(12);
648      }
649      return retornaCabezaP(pila.lista);
650  }
651
652
653  /*destruye la pila*/
654  void destruirPila(struct Pila pila){
655      destruir(pila.lista);
656  }
657
658  /*Recordar que las Pilas no se recorren en forma secuencial*/
659  /*Se va utilizar solo para mostrar los valores*/
660  void imprimir(const struct Pila & pila){
661      imprimeP(pila.lista);
662  }
663
664  ///*imprime desapilando*/
665  //void imprime(struct Pila & pila){
666  //
667  //    while(not esPilaVacia(pila)){
668  //        cout<<cima(pila)<<"-";
669  //        desapilar(pila);
670  //    }
671  //}
672
673  /*
674   * File:   funcionesArbolesBB.cpp
675   * Author: ANA RONCAL
676   * Created on 19 de septiembre de 2023, 10:46 AM
677   */
678
679  #include <iostream>
680  #include <iomanip>
681  #include <fstream>
682  #include <cstring>
683  #include "NodoArbol.h"
684  #include "Nodo.h"
685  #include "ArbolBinario.h"
686  #include "Lista.h"
687  #include "ListaP.h"
688  #include "Cola.h"
689  #include "Pila.h"
690  using namespace std;
```

```
691  #include "funcionesArbolesBinarios.h"
692  #include "funcionesCola.h"
693  #include "funcionesPila.h"
694
695  void construir(struct ArbolBinario & arbol ){
696      arbol.raiz = nullptr;
697  }
698
699  bool esNodoVacio(struct NodoArbol * nodo){
700      return nodo == nullptr;
701  }
702
703  bool esArbolVacio( struct ArbolBinario arbol){
704      return esNodoVacio(arbol.raiz);
705  }
706
707  struct NodoArbol * crearNuevoNodoArbol(struct NodoArbol * arbolIzquierdo,
708                                  int elemento, struct NodoArbol * arbolDerecho){
709      struct NodoArbol * nuevoNodo = new struct NodoArbol;
710      nuevoNodo->elemento = elemento;
711      nuevoNodo->izquierda = arbolIzquierdo;
712      nuevoNodo->derecha = arbolDerecho;
713      return nuevoNodo;
714  }
715
716  void plantarArbolBinario(struct ArbolBinario & arbol, struct NodoArbol * arbolIzquierdo,
717                           int elemento, struct NodoArbol * arbolDerecho){
718
719      struct NodoArbol * nuevoNodo = crearNuevoNodoArbol(arbolIzquierdo, elemento,
         arbolDerecho);
720      arbol.raiz = nuevoNodo;
721  }
722
723  void plantarArbolBinario(struct ArbolBinario & arbol, struct ArbolBinario arbolIzquierdo,
724                           int elemento, struct ArbolBinario arbolDerecho){
725
726      struct NodoArbol * nuevoNodo = crearNuevoNodoArbol(arbolIzquierdo.raiz, elemento,
         arbolDerecho.raiz);
727      arbol.raiz = nuevoNodo;
728  }
729
730  int raiz(struct NodoArbol * nodo){
731      if (esNodoVacio(nodo)){
732          cout<<"No se puede obtener raíz de un árbol vacio"<<endl;
733          exit(1);
734      }
735      return nodo->elemento;
736  }
737
738  struct NodoArbol * hijoDerecho(struct ArbolBinario arbol){
739      if (esArbolVacio(arbol)){
740          cout<<"No se puede obtener raíz de un árbol vacio"<<endl;
741          exit(1);
742      }
743      return arbol.raiz->derecha;
744  }
745
746  struct  NodoArbol * hijoIzquierdo(struct ArbolBinario arbol){
747      if (esArbolVacio(arbol)){
748          cout<<"No se puede obtener raíz de un árbol vacio"<<endl;
749          exit(1);
750      }
751
752      return arbol.raiz->izquierda;
753   }
754
755  void imprimeRaiz(struct ArbolBinario arbol){
756      imprimeNodo(arbol.raiz);
757  }
```

```
758
759 void imprimeNodo(struct NodoArbol * nodo){
760     cout<<setw(5)<<nodo->elemento;
761 }
762
763 void recorrerEnOrdenRecursivo(struct NodoArbol * nodo){
764     if(not esNodoVacio(nodo)){
765         recorrerEnOrdenRecursivo(nodo->izquierda);
766         imprimeNodo(nodo);
767         recorrerEnOrdenRecursivo(nodo->derecha);
768     }
769 }
770
771 /*En árbol, se lleva a cabo visitando el hijo izquierdo del nodo, luego el nodo
772  luego todos los restantes, comenzando por la raíz*/
773 void recorrerEnOrden(struct ArbolBinario arbol){
774     /*Imprime en orden*/
775     if (not esArbolVacio(arbol)){
776         recorrerEnOrdenRecursivo(arbol.raiz);
777     }
778 }
779
780 void recorrerEnPreOrdenRecursivo(struct NodoArbol * nodo){
781     if(not esNodoVacio(nodo)){
782         imprimeNodo(nodo);
783         recorrerEnPreOrdenRecursivo(nodo->izquierda);
784         recorrerEnPreOrdenRecursivo(nodo->derecha);
785     }
786 }
787
788 /*recorrido descendente, se lleva a cabo visitando cada nodo, seguido de sus hijos,
789  luego todos los restantes, comenzando por la raíz*/
790 void recorrerPreOrden(struct ArbolBinario arbol){
791     if (not esArbolVacio(arbol)){
792         recorrerEnPreOrdenRecursivo(arbol.raiz);
793     }
794 }
795
796 void recorrerEnPostOrdenRecursivo(struct NodoArbol * nodo){
797     if(not esNodoVacio(nodo)){
798         recorrerEnPostOrdenRecursivo(nodo->izquierda);
799         recorrerEnPostOrdenRecursivo(nodo->derecha);
800         imprimeNodo(nodo);
801     }
802 }
803
804 /*recorrido ascendente, se lleva a cabo visitando los hijos, y luego el nodo
805  luego todos los restantes, comenzando por la raíz*/
806 void recorrerPostOrden(struct ArbolBinario arbol){
807
808     if (not esArbolVacio(arbol)){
809         recorrerEnPostOrdenRecursivo(arbol.raiz);
810     }
811 }
812
813 int maximo(int a, int b){
814     return a>=b ? a: b;
815 }
816
817 int alturaRecursivo(struct NodoArbol * nodo){
818     if(esNodoVacio(nodo))
819         return 0;
820     else if(esNodoVacio(nodo->izquierda) and esNodoVacio(nodo->derecha))
821         return 0;
822     else
823         return 1 + maximo( alturaRecursivo(nodo->izquierda), alturaRecursivo(nodo->
        derecha));
824 }
825
```

```
826 int altura(struct ArbolBinario arbol){
827     return alturaRecursivo(arbol.raiz); //como el arbol ha sido construido no va apuntar
        a nullptr
828 }
829
830 int numeroNodosRecursivo(struct NodoArbol * nodo){
831     if(esNodoVacio(nodo))
832         return 0;
833     else
834         return 1 + numeroNodosRecursivo(nodo->izquierda) + numeroNodosRecursivo(nodo->
            derecha);
835 }
836
837 /*Determina el número de elementos del árbol*/
838 int numeroNodos(struct ArbolBinario arbol){
839     return numeroNodosRecursivo(arbol.raiz);
840 }
841
842 int numeroHojasRecursivo(struct NodoArbol * nodo){
843     if(esNodoVacio(nodo))
844         return 0;
845     else if ( esNodoVacio(nodo->izquierda) and esNodoVacio(nodo->derecha) )
846         return 1;
847     else
848         return numeroHojasRecursivo(nodo->izquierda) + numeroHojasRecursivo(nodo->derecha
            );
849 }
850
851 int numeroHojas(struct ArbolBinario arbol){
852     return numeroHojasRecursivo(arbol.raiz);
853 }
854
855 int esEquilibradoRecursivo(struct NodoArbol * nodo){
856     if(esNodoVacio(nodo))
857         return 1;
858     else{
859         int alturaHijoIzquierdo = alturaRecursivo(nodo->izquierda);
860         int alturaHijoDerecho = alturaRecursivo(nodo->derecha);
861         int diferencia = abs(alturaHijoIzquierdo - alturaHijoDerecho);
862         return diferencia<=1 and
863                esEquilibradoRecursivo(nodo->izquierda) and
864                esEquilibradoRecursivo(nodo->derecha);
865     }
866 }
867
868 int esEquilibrado(struct ArbolBinario arbol ){
869     return esEquilibradoRecursivo(arbol.raiz);
870 }
871
872 int esHoja(struct ArbolBinario arbol){
873     if(esArbolVacio(arbol))
874         return 0;
875     else
876         return esNodoVacio(arbol.raiz->izquierda) and esNodoVacio(arbol.raiz->derecha);
877 }
878
879 void destruirArbolBinario(struct ArbolBinario arbol){
880     destruirRecursivo(arbol.raiz);
881     arbol.raiz = nullptr;
882 }
883
884 void destruirRecursivo(struct NodoArbol * nodo){
885     if(not (esNodoVacio(nodo))){
886         destruirRecursivo(nodo->izquierda);
887         destruirRecursivo(nodo->derecha);
888         delete nodo;
889         nodo = nullptr;
890     }
891 }
```

```
892
893 /*recorre el árbol por niveles usando una cola*/
894 void recorridoPorNivel(struct ArbolBinario arbol){
895     struct Cola cola; /*Se usa una cola para acceder a los nodos*/
896     construir(cola);
897     if(not esArbolVacio(arbol)){
898         encolar(cola, arbol.raiz);
899         while(not esColaVacia(cola)){
900             struct NodoArbol * nodo = desencolar(cola);
901             imprimeNodo(nodo);
902             if (not esNodoVacio(nodo->izquierda)){
903                 encolar(cola, nodo->izquierda);
904             }
905             if (not esNodoVacio(nodo->derecha)){
906                 encolar(cola, nodo->derecha);
907             }
908         }
909     }
910     destruirCola(cola);
911 }
912
913 void enOrdenIterativo(struct ArbolBinario arbol){
914     struct Pila pila; /*Se usa una pila para acceder a los nodos*/
915     construir(pila);
916     int fin = 0;
917     do{
918         while (not esArbolVacio(arbol)){
919             apilar(pila, arbol.raiz);
920             arbol.raiz = arbol.raiz->izquierda;
921         }
922         if (esPilaVacia(pila))
923             fin = 1;
924         else{
925             arbol.raiz = desapilar(pila);
926             imprimeRaiz(arbol);
927             arbol.raiz = arbol.raiz->derecha;
928         }
929     } while(fin == 0);
930     destruirPila(pila);
931 }
932
933 void preOrdenIterativo(struct ArbolBinario arbol){
934     struct Pila pila;  /*Se usa una pila para acceder a los nodos*/
935     construir(pila);
936     if (not esArbolVacio(arbol)){
937         apilar(pila, arbol.raiz);
938         while(not esPilaVacia(pila)){
939             struct NodoArbol * nodo = desapilar(pila);
940             imprimeNodo(nodo);
941             if (not esNodoVacio(nodo->derecha))
942                 apilar(pila, nodo->derecha);
943             if (not esNodoVacio(nodo->izquierda))
944                 apilar(pila, nodo->izquierda);
945         }
946     }
947     destruirPila(pila);
948 }
949
```