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

#include <iostream>
#include <cstdlib>
#include "ArbolBinario.h"
#include "ArbolBinarioBusqueda.h"
using namespace std;
#include "funcionesArbolesBB.h"
#include "funcionesArbolesBinarios.h"

/*
 * IMPLEMENTA ARBOLES DE BÚSQUEDA BINARIA
 */
int main(int argc, char** argv) {

    struct ArbolBinarioBusqueda arbol, arbol2;
    int auxiliarElemento;

    construir(arbol);
    insertar(arbol, 100);
    insertar(arbol, 50);
    insertar(arbol, 200);
    insertar(arbol, 25);
    insertar(arbol, 75);

    preOrden(arbol);
    enOrden(arbol);
    postOrden(arbol);

    construir(arbol2);
    insertar(arbol2, 5);
    insertar(arbol2, 15);
    insertar(arbol2, 3);
    insertar(arbol2, 7);
    insertar(arbol2, 10);

    preOrden(arbol2);
    enOrden(arbol2);
    postOrden(arbol2);

    auxiliarElemento = minimoABB(arbol);
    cout<<"Mínimo ABB: "<<auxiliarElemento<<endl;

    auxiliarElemento = maximoABB(arbol);
    cout<<"Máximo ABB: "<<auxiliarElemento<<endl;

    cout<<"Sumar nodos: "<<sumarNodos(arbol)<<endl;

    borraNodo(arbol, 100);
    recorridoPorNivel(arbol);
    enOrden(arbol);

    destruirArbolBB(arbol);
    destruirArbolBB(arbol2);


    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 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:   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:   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:   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:   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 &);
void imprimeP( struct Lista );

#endif /* FUNCIONESLISTA_H */

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

#ifndef FUNCIONESLISTA_H
#define FUNCIONESLISTA_H

```

```
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
```

```
#endif /* ARBOLBINARIOBUSQUEDA_H */

/*
 * File:   funcionesArboles.h
 * Author: ANA RONCAL
 * Created on 18 de septiembre de 2023, 06:00 PM
 */

#ifndef FUNCIONESARBOLES_H
#define FUNCIONESARBOLES_H

void construir(struct ArbolBinario & );

bool esArbolVacio(struct ArbolBinario arbol);
bool esNodoVacio(struct NodoArbol * nodo);

struct NodoArbol * crearNuevoNodoArbol(struct NodoArbol *, int,
                                                    struct NodoArbol *);
void plantarArbolBinario(struct ArbolBinario &, struct NodoArbol *, int,
                                                  struct NodoArbol * );
void plantarArbolBinario(struct ArbolBinario &, struct ArbolBinario, int,
                                                  struct ArbolBinario );
int raiz(struct NodoArbol * nodo);
void imprimeRaiz(struct ArbolBinario arbol);
void imprimeNodo(struct NodoArbol * nodo);

struct  NodoArbol * hijoDerecho(struct ArbolBinario );
struct  NodoArbol * hijoIzquierdo(struct ArbolBinario );

void recorrerEnPreOrdenRecursivo(struct NodoArbol * nodo);
void recorrerEnOrdenRecursivo(struct NodoArbol * nodo);
void recorrerEnPostOrdenRecursivo(struct NodoArbol * nodo);

void recorrerEnOrden(struct ArbolBinario );
void recorrerPreOrden(struct ArbolBinario );
void recorrerPostOrden(struct ArbolBinario );

int altura(struct ArbolBinario);
int alturaRecursivo(struct NodoArbol * nodo);

int numeroNodos(struct ArbolBinario );
int numeroNodosRecursivo(struct NodoArbol * nodo);

int numeroHojas(struct ArbolBinario );

int sumarNodosRecursivo(struct NodoArbol * nodo);
int sumarNodos(struct ArbolBinario arbol);

int esEquilibrado(struct ArbolBinario );
int esEquilibradoRecursivo(struct NodoArbol * nodo);

int esHoja(struct ArbolBinario );

void destruirArbolBinario(struct ArbolBinario );
void destruirRecursivo(struct NodoArbol *);

void recorridoPorNivel(struct ArbolBinario arbol);
void enOrdenIterativo(struct ArbolBinario arbol);
void preOrdenIterativo(struct ArbolBinario arbol);


#endif /* FUNCIONESARBOLES_H */

/*
 * File:   funcionesArbolesBB.h
 * Author: ANA RONCAL
 * Created on 24 de septiembre de 2023, 10:34 PM
 */
```

```
#ifndef FUNCIONESARBOLESBB_H
#define FUNCIONESARBOLESBB_H

void construir(struct ArbolBinarioBusqueda & arbol);
void insertar(struct ArbolBinarioBusqueda & arbol, int elemento);
bool esArbolVacio(struct ArbolBinarioBusqueda  arbol);
void insertarRecursivo(struct NodoArbol *& raiz, int elemento);
void plantarArbolBB(struct NodoArbol *& arbol,
                        struct NodoArbol * arbolIzquierdo, int elemento,
                        struct NodoArbol * arbolDerecho);
void preOrden(struct ArbolBinarioBusqueda arbol);
void enOrden(struct ArbolBinarioBusqueda arbol);
void postOrden(struct ArbolBinarioBusqueda arbol);

void destruirArbolBB(struct ArbolBinarioBusqueda arbol);

int minimoABB(struct ArbolBinarioBusqueda );
int minimoABBRecursivo(struct NodoArbol * nodo);
int maximoABBRecursivo(struct NodoArbol * nodo);
int maximoABB(struct ArbolBinarioBusqueda );
struct NodoArbol *  minimoArbol(struct NodoArbol * nodo);

bool buscaArbolRecursivo(struct NodoArbol * nodo, int dato);
bool buscaArbol(struct ArbolBinarioBusqueda arbol ,int dato);

void borraNodo(struct ArbolBinarioBusqueda,int );
struct NodoArbol * borraNodoRecursivo(struct NodoArbol * nodo, int elemento);
int comparaABB(int , int );

int sumarNodos(struct ArbolBinarioBusqueda arbol);

void recorridoPorNivel(struct ArbolBinarioBusqueda );

#endif /* FUNCIONESARBOLESBB_H */



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

#include <iostream>
#include "ArbolBinario.h"
#include "Nodo.h"
#include "NodoArbol.h"
#include "Lista.h"
using namespace std;
#include "funcionesLista.h"
#include "funcionesArbolesBinarios.h"
#include "Nodo.h"
#include "Lista.h"


/*Valores iniciales de la lista*/
void construir(struct Lista & lista){
    lista.cabeza = nullptr;
    lista.cola = nullptr;
    lista.longitud = 0;
}

/*devuelve si la lista esta vacia 1, caso contrario 0 */
bool esListaVacia( struct Lista  lista){
    return lista.cabeza == nullptr;
}

/*DEVUELVE LA CANTIDAD DE ELEMENTOS DE LA LISTA*/
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 estÃ¡ vacÃ-a";
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¡ vacÃ-a";
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 está vacía"<<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 después 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"
```

```
#include "ListaP.h"
#include "Pila.h"
using namespace std;
#include "funcionesListaP.h"
#include "funcionesPila.h"

/*constructor de Pila*/
void construir(struct Pila & pila){
    construirP(pila.lista);
}

/*Determina si la pila está vacía*/
bool esPilaVacia( struct Pila  pila){
    return esListaVaciaP(pila.lista);
}

/*Determina el número de elementos de la pila*/
int longitud( struct Pila   pila){
    return longitudP(pila.lista);
}

/*push, añade un elemento a la parte superior de la pila*/
void apilar(struct Pila & pila, struct NodoArbol * nodo){
    insertarAlInicio(pila.lista, nodo);
}

/*pop, elimina un elemento de la parte superior de la pila*/
struct NodoArbol * desapilar(struct Pila & pila){
    if (esPilaVacia(pila)){
        cout<<"La pila está vacía, por lo tanto no se puede desapilar"<<endl;
    exit(11);
    }
    struct NodoArbol * nodo = cima(pila);
    eliminaCabezaP(pila.lista);
    return nodo;
}

/*examina un elemento situado en la parte superior de la pila*/
struct NodoArbol * cima(struct Pila pila){
    if (esPilaVacia(pila)){
    cout<<"La pila está vacía por lo tanto no posee cima"<<endl;
    exit(12);
    }
    return retornaCabezaP(pila.lista);
}


/*destruye la pila*/
void destruirPila(struct Pila pila){
    destruir(pila.lista);
}

/*Recordar que las Pilas no se recorren en forma secuencial*/
/*Se va utilizar solo para mostrar los valores*/
void imprimir(const struct Pila & pila){
    imprimeP(pila.lista);
}

///*imprime desapilando*/
//void imprime(struct Pila & pila){
//
//    while(not esPilaVacia(pila)){
//        cout<<cima(pila)<<"-";
//        desapilar(pila);
//    }
//}

/*
 * File:   funcionesCola.cpp
```

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

#include <iostream>
#include <iomanip>
#include "ArbolBinario.h"
#include "Nodo.h"
#include "Lista.h"
#include "Cola.h"
using namespace std;
#include "funcionesLista.h"
#include "funcionesCola.h"


#define PROCESO 120
#define MAX_CAJEROS 10
#define NUM_CLIENTES 100

void construir(struct Cola & cola){
    construir(cola.lista);
}

bool esColaVacia( struct Cola  cola){
    return esListaVacia(cola.lista);
}

int longitud(struct Cola cola){
    return longitud(cola.lista);
}

void encolar(struct Cola & cola, struct NodoArbol * nodo){
    insertarAlFinal(cola.lista, nodo);
}

struct NodoArbol *  desencolar(struct Cola & cola){
    if(esColaVacia(cola)){
        cout<<"La cola está vacía no se puede desencolar"<<endl;
        exit(1);
    }

    struct NodoArbol * nodo = retornaCabeza(cola.lista);
    eliminaCabeza(cola.lista);
    return nodo;
}

void destruirCola(struct Cola & cola){
    destruirLista(cola.lista);
}

void imprime( struct Cola  cola){
    imprime(cola.lista);
}

/*
 * File:   funcionesArbolesBB.cpp
 * Author: ANA RONCAL
 * Created on 19 de septiembre de 2023, 10:46 AM
 */

#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstring>
#include "NodoArbol.h"
#include "Nodo.h"
#include "ArbolBinario.h"
#include "Lista.h"
#include "ListaP.h"
```

```cpp
#include "Cola.h"
#include "Pila.h"
using namespace std;
#include "funcionesArbolesBinarios.h"
#include "funcionesCola.h"
#include "funcionesPila.h"

void construir(struct ArbolBinario & arbol ){
    arbol.raiz = nullptr;
}

bool esNodoVacio(struct NodoArbol * nodo){
    return nodo == nullptr;
}

bool esArbolVacio( struct ArbolBinario arbol){
    return esNodoVacio(arbol.raiz);
}

struct NodoArbol * crearNuevoNodoArbol(struct NodoArbol * arbolIzquierdo,
                                int elemento, struct NodoArbol * arbolDerecho){
    struct NodoArbol * nuevoNodo = new struct NodoArbol;
    nuevoNodo->elemento = elemento;
    nuevoNodo->izquierda = arbolIzquierdo;
    nuevoNodo->derecha = arbolDerecho;
    return nuevoNodo;
}

void plantarArbolBinario(struct ArbolBinario & arbol, struct NodoArbol * arbolIzquierdo,
                         int elemento, struct NodoArbol * arbolDerecho){

    struct NodoArbol * nuevoNodo = crearNuevoNodoArbol(arbolIzquierdo, elemento,
    arbolDerecho);
    arbol.raiz = nuevoNodo;
}

void plantarArbolBinario(struct ArbolBinario & arbol, struct ArbolBinario arbolIzquierdo,
                         int elemento, struct ArbolBinario arbolDerecho){

    struct NodoArbol * nuevoNodo = crearNuevoNodoArbol(arbolIzquierdo.raiz, elemento,
    arbolDerecho.raiz);
    arbol.raiz = nuevoNodo;
}

int raiz(struct NodoArbol * nodo){
    if (esNodoVacio(nodo)){
        cout<<"No se puede obtener raíz de un árbol vacio"<<endl;
        exit(1);
    }
    return nodo->elemento;
}

struct NodoArbol * hijoDerecho(struct ArbolBinario arbol){
    if (esArbolVacio(arbol)){
        cout<<"No se puede obtener raíz de un árbol vacio"<<endl;
        exit(1);
    }
    return arbol.raiz->derecha;
}

struct  NodoArbol * hijoIzquierdo(struct ArbolBinario arbol){
    if (esArbolVacio(arbol)){
        cout<<"No se puede obtener raíz de un árbol vacio"<<endl;
        exit(1);
    }

    return arbol.raiz->izquierda;
 }

```

```
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);
```

```
            if (not esNodoVacio(nodo->derecha))
                apilar(pila, nodo->derecha);
            if (not esNodoVacio(nodo->izquierda))
                apilar(pila, nodo->izquierda);
        }
    }
    cout<<endl;
    destruirPila(pila);
}

/*
 * File:   funcionesArbolesBB.cpp
 * Author: ANA RONCAL
 * Created on 24 de septiembre de 2023, 10:36 PM
 */

#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstring>
#include "ArbolBinario.h"
#include "funcionesArbolesBB.h"
#include "ArbolBinarioBusqueda.h"
using namespace std;
#include "funcionesArbolesBinarios.h"
#include "NodoArbol.h"

void construir(struct ArbolBinarioBusqueda & arbol){
    construir(arbol.arbolBinario);
}

bool esArbolVacio(struct ArbolBinarioBusqueda  arbol){
    esArbolVacio(arbol.arbolBinario);
}

void plantarArbolBB(struct NodoArbol *& arbol,
                        struct NodoArbol * arbolIzquierdo, int elemento,
                        struct NodoArbol * arbolDerecho){

    struct NodoArbol * nuevoNodo = new struct NodoArbol;
    nuevoNodo->elemento = elemento;
    nuevoNodo->izquierda = arbolIzquierdo;
    nuevoNodo->derecha = arbolDerecho;
    arbol = nuevoNodo;
}
void insertar(struct ArbolBinarioBusqueda & arbol, int elemento){
    insertarRecursivo(arbol.arbolBinario.raiz, elemento);
}

void insertarRecursivo(struct NodoArbol *& raiz, int elemento){
    if(raiz == nullptr)
        plantarArbolBB(raiz, nullptr, elemento, nullptr);
    else
        if (raiz->elemento > elemento)
            insertarRecursivo(raiz->izquierda, elemento);
        else
            insertarRecursivo(raiz->derecha, elemento);
}

void enOrden(struct ArbolBinarioBusqueda arbol){
    recorrerEnOrden(arbol.arbolBinario);
}

void preOrden(struct ArbolBinarioBusqueda arbol){
    recorrerPreOrden(arbol.arbolBinario);
}

void postOrden(struct ArbolBinarioBusqueda arbol){
    recorrerPostOrden(arbol.arbolBinario);
```

```
}

void destruirArbolBB(struct ArbolBinarioBusqueda arbol){
    destruirArbolBinario(arbol.arbolBinario);
}

int minimoABBRecursivo(struct NodoArbol * nodo){
    if(esNodoVacio(nodo)){
        cout<<"El árbol está vacío"<<endl;
        exit(1);
    }
    if(esNodoVacio(nodo->izquierda))
        return nodo->elemento;
    minimoABBRecursivo(nodo->izquierda);
}

int minimoABB(struct ArbolBinarioBusqueda arbol){
    return minimoABBRecursivo(arbol.arbolBinario.raiz);
}

int maximoABBRecursivo(struct NodoArbol * nodo){
    if(esNodoVacio(nodo)){
        cout<<"El árbol está vacío"<<endl;
        exit(1);
    }
    if(esNodoVacio(nodo->derecha))
        return nodo->elemento;
    maximoABBRecursivo(nodo->derecha);
}

int maximoABB(struct ArbolBinarioBusqueda arbol){
    return maximoABBRecursivo(arbol.arbolBinario.raiz);
}
struct NodoArbol *  minimoArbol(struct NodoArbol * nodo){
    if(esNodoVacio(nodo))
        return nodo;
    if(esNodoVacio(nodo->izquierda))
        return nodo;
    minimoArbol(nodo->izquierda);
}

struct NodoArbol * borraNodoRecursivo(struct NodoArbol * nodo, int dato){
    if(esNodoVacio(nodo))
        return nodo;
    if(comparaABB(nodo->elemento, dato) == 1){ //nodo mayor que el dato
        nodo->izquierda = borraNodoRecursivo(nodo->izquierda, dato);
    }
    else{
        if(comparaABB(nodo->elemento, dato) == -1){ //nodo menor que el dato
            nodo->derecha = borraNodoRecursivo(nodo->derecha, dato);
        }
        else{
            if(esNodoVacio(nodo->izquierda)){
                struct NodoArbol * elimina = nodo->derecha;
                delete elimina;
                return elimina;
            }
            else{
                if(esNodoVacio(nodo->derecha)){
                    struct NodoArbol * elimina = nodo->izquierda;
                    delete elimina;
                    return elimina;
                }
                else{
                    struct NodoArbol * temp = minimoArbol(nodo->derecha);
                    nodo->elemento = temp->elemento;
                    nodo->derecha = borraNodoRecursivo(nodo->derecha, temp->elemento);
                }
            }
```

```
        }
    }
    return nodo;
}

void  borraNodo(struct ArbolBinarioBusqueda arbol, int dato){
    arbol.arbolBinario.raiz = borraNodoRecursivo(arbol.arbolBinario.raiz, dato);
}

int comparaABB(int elementoA, int elementoB ){
    if(elementoA == elementoB) return 0;
    else if(elementoA < elementoB) return -1;
    else if (elementoA > elementoB) return 1;
}

bool buscaArbolRecursivo(struct NodoArbol * nodo, int dato){
    if(esNodoVacio(nodo)){
        return false;
    }
    if(comparaABB(nodo->elemento, dato) == 0)
        return true;
    if(comparaABB(nodo->elemento, dato) == 1)
        return buscaArbolRecursivo(nodo->izquierda, dato);
    else
        if(comparaABB(nodo->elemento, dato) == -1)
            return buscaArbolRecursivo(nodo->derecha, dato);
}

bool buscaArbol(struct ArbolBinarioBusqueda arbol ,int dato){
    return buscaArbolRecursivo(arbol.arbolBinario.raiz, dato);
}
void recorridoPorNivel(struct ArbolBinarioBusqueda arbol){
    recorridoPorNivel(arbol.arbolBinario);
}

int sumarNodos(struct ArbolBinarioBusqueda arbol){
    return sumarNodos(arbol.arbolBinario);
}
```