```
#include <stdio.h>
#include <stdlib.h>

// Estructura del nodo
typedef struct _nodo {
   int dato;
   struct _nodo *izquierdo;
   struct _nodo *derecho;
} tipoNodo;

// Definiciones de tipo
typedef tipoNodo *pNodo;
typedef tipoNodo *Arbol;

// Prototipos de funciones
void Insertar(Arbol *a, int dat);
void Borrar(Arbol *a, int dat);
int Buscar(Arbol a, int dat);
void Mostrar(Arbol a);
void InOrden(Arbol a);
void Actualizar(Arbol *a, int viejoDato, int nuevoDato);
void LiberarArbol(Arbol a);
pNodo BuscarSucesor(pNodo a);

// Función principal
int main() {
   Arbol miArbol = NULL;
   int opcion, dato, viejoDato, nuevoDato;
```

```
do {
    // Menú
    printf("\n--- Menú de Operaciones ---\n");
    printf("1. Insertar\n");
    printf("2. Borrar\n");
    printf("3. Buscar\n");
    printf("4. Mostrar (inorden)\n");
    printf("5. Actualizar\n");
    printf("6. Salir\n");
    printf("Seleccione una opción: ");
    scanf("%d", &opcion);

    switch (opcion) {
        case 1:
            printf("Ingrese el dato a insertar: ");
            scanf("%d", &dato);
            Insertar(&miArbol, dato);
            break;
        case 2:
            printf("Ingrese el dato a borrar: ");
            scanf("%d", &dato);
            Borrar(&miArbol, dato);
            break;
        case 3:
            printf("Ingrese el dato a buscar: ");
            scanf("%d", &dato);
            if (Buscar(miArbol, dato)) {
                printf("%d encontrado en el árbol.\n", dato);
            } else {
```

```
            printf("%d no encontrado en el árbol.\n", dato);
        }
        break;
    case 4:
        printf("Árbol en inorden: ");
        InOrden(miArbol);
        printf("\n");
        break;
    case 5:
        printf("Ingrese el dato a actualizar: ");
        scanf("%d", &viejoDato);
        printf("Ingrese el nuevo dato: ");
        scanf("%d", &nuevoDato);
        Actualizar(&miArbol, viejoDato, nuevoDato);
        break;
    case 6:
        printf("Saliendo del programa. ¡Hasta luego!\n");
        break;
    default:
        printf("Opción no válida. Inténtelo de nuevo.\n");
  }

} while (opcion != 6);

// Libera la memoria al finalizar el programa
// (puedes implementar una función para liberar el árbol si es más complejo)
LiberarArbol(miArbol);

return 0;
```

```
}

// Implementación de las funciones

void Insertar(Arbol *a, int dat) {
    if (*a == NULL) {
        // Crear un nuevo nodo
        *a = (Arbol)malloc(sizeof(tipoNodo));
        (*a)->dato = dat;
        (*a)->izquierdo = NULL;
        (*a)->derecho = NULL;
    } else {
        // Determinar si el nuevo dato va a la izquierda o a la derecha
        if (dat < (*a)->dato) {
            Insertar(&((*a)->izquierdo), dat);
        } else if (dat > (*a)->dato) {
            Insertar(&((*a)->derecho), dat);
        }
        // Ignoramos el caso en que el dato ya exista en el árbol
    }
}

void Borrar(Arbol *a, int dat) {
    if (*a != NULL) {
        if (dat < (*a)->dato) {
            Borrar(&((*a)->izquierdo), dat);
        } else if (dat > (*a)->dato) {
            Borrar(&((*a)->derecho), dat);
        } else {
```

```
            // Encontramos el nodo a borrar
            pNodo tmp = *a;
            if ((*a)->izquierdo == NULL) {
               *a = (*a)->derecho;
            } else if ((*a)->derecho == NULL) {
               *a = (*a)->izquierdo;
            } else {
               // El nodo tiene dos hijos, buscamos el sucesor inorden
               tmp = BuscarSucesor((*a)->derecho);
               (*a)->dato = tmp->dato;
               Borrar(&((*a)->derecho), tmp->dato);
               return;
            }
            free(tmp);
         }
      }
   }

int Buscar(Arbol a, int dat) {
   if (a == NULL) {
      return 0;
   } else if (dat == a->dato) {
      return 1;
   } else if (dat < a->dato) {
      return Buscar(a->izquierdo, dat);
   } else {
      return Buscar(a->derecho, dat);
   }
}
```

```
void Mostrar(Arbol a) {
   printf("Árbol en inorden: ");
   InOrden(a);
   printf("\n");
}

void InOrden(Arbol a) {
   if (a != NULL) {
      InOrden(a->izquierdo);
      printf("%d ", a->dato);
      InOrden(a->derecho);
   }
}

void Actualizar(Arbol *a, int viejoDato, int nuevoDato) {
   Borrar(a, viejoDato);
   Insertar(a, nuevoDato);
}

pNodo BuscarSucesor(pNodo a) {
   while (a->izquierdo != NULL) {
      a = a->izquierdo;
   }
   return a;
}

void LiberarArbol(Arbol a) {
   if (a != NULL) {
```

```
        LiberarArbol(a->izquierdo);
        LiberarArbol(a->derecho);
        free(a);
    }
}
```