```
#include <iostream>
#include <cstdlib>
using namespace std;

struct Nodo{
	int info;
	Nodo *right;
	Nodo *left;
	Nodo *daddy;
};

void opcionMenu(Nodo*&);
void menu();

Nodo *crearNodo(int, Nodo*);
void insertarNodo(Nodo*&, int, Nodo*);
void mostrarArbol(Nodo*, int);
bool busqueda(Nodo*, int);

void menuRecorrer(Nodo*);
void preOrden(Nodo*);
void inOrden(Nodo*);
void postOrden(Nodo*);

void eliminar(Nodo*&, int);
void eliminarNodo(Nodo*&);
Nodo *minimo(Nodo*);
void reemplazar(Nodo*&, Nodo*&);
void destruirNodo(Nodo*&);

int main(){
	Nodo *root = NULL;
	opcionMenu(root);
	return 0;
}

void menu(){
	cout << " //         .:Menu:.         //\n";
	cout << " a) Insertar un nodo en el arbol.\n";
	cout << " b) Mostrar el arbol completo.\n";
	cout << " c) Buscar un nodo especifico.\n";
	cout << " d) Recorrer el arbol.\n";
	cout << " e) Borrar un nodo del arbol.\n";
	cout << " f) Salir.\n";
}

void opcionMenu(Nodo *&arbol){
	menu();
	bool flag = true;
	char op;
	int dato;
	while (flag){
		cout << " >Opcion: ";
		cin >> op;
		switch(op){
			case 'a':
				cout << "\tDigite un numero: ";
				cin >> dato;
				insertarNodo(arbol, dato, NULL);
			break;
```

«***include***» Directiva de preprocesador que se usa en los lenguajes para «incluir» las declaraciones de otro fichero en la compilación.

«***NULL***» Esta macro se expande a una constante de puntero nulo.

«***return***» Finaliza la función y devuelve el control al punto donde fue llamada.

«***cout***» El flujo de salida estándar es el destino por defecto de los caracteres determinados por el entorno.

«***cin***» El flujo de entrada estándar es una fuente de caracteres determinada por el entorno. Por lo general, se supone que la entrada proviene de una fuente externa, como el teclado o un archivo.

```
                case 'b':
                        cout << "Mostrando arbol completo...\n\n";
                        mostrarArbol(arbol, 0);
                        cout << "\n";
                        system("pause");
                        system("cls");
                        menu();
                break;

                case 'c':
                        cout << "Digite el elemento a buscar: ";
                        cin >> dato;
                        if (busqueda(arbol, dato)){
                                cout << "\tElemento '" << dato << "' encontrado.\n";
                        }else{
                                cout << "\tElemento no encontrado.\n";
                        }
                break;

                case 'd':
                        menuRecorrer(arbol);
                        cout << "\n";
                        system("pause");
                        system("cls");
                        menu();
                break;

                case 'e':
                        cout << "\tDigite el número a eliminar: ";
                        cin >> dato;
                        eliminar(arbol, dato);
                break;

                case 'f':
                        cout << "\tMoi, moi!\n";
                        flag = false;
                break;

                default:
                        cout << "\tOpcion no valida.\n";
                break;
        }
    }
}

Nodo *crearNodo(int n, Nodo *padre){
    Nodo *new_node = new Nodo();

    new_node->info = n;
    new_node->right = NULL;
    new_node->left = NULL;
    new_node->daddy = padre;

    return new_node;
}

void insertarNodo(Nodo *&arbol, int n, Nodo *padre){
    if (arbol == NULL){
        Nodo *new_node = crearNodo(n, padre);
```

«***system***» Invoca el procesador de comandos para ejecutar un comando.

«***new***» Esta cabecera describe las funciones utilizadas para gestionar el almacenamiento dinámico en C++.

```
			arbol = new_node;
		}else{
			int valorRaiz = arbol->info;
			if (n < valorRaiz){
				insertarNodo(arbol->left, n, arbol);
			}else{
				insertarNodo(arbol->right, n, arbol);
			}
		}
}

void mostrarArbol(Nodo *arbol, int cont){
		if (arbol == NULL){
			return;
		}else{
			mostrarArbol(arbol->right, cont+1);
			for (int i = 0; i < cont; i++){
				cout << "   ";
			}
			cout << arbol->info << endl;
			mostrarArbol(arbol->left, cont+1);
		}
}

bool busqueda(Nodo *arbol, int n){
		if (arbol == NULL){
			return false;
		}else if (arbol->info == n){
			return true;
		}else{
			if(n < arbol->info){
				return busqueda(arbol->left, n);
			}else{
				return busqueda(arbol->right, n);
			}
		}
}

void menuRecorrer(Nodo *arbol){
		cout << "   .:Menu Recorrer:. \n";
		cout << "   1) Pre-Orden.\n";
		cout << "   2) In-Orden.\n";
		cout << "   3) Post-Orden.\n";

		cout << "   > ";
		char ac;
		cin >> ac;

		switch(ac){
			case '1':
				cout << "\tRecorrido Pre-Orden:\n  ";
				preOrden(arbol);
			break;

			case '2':
				cout << "\tRecorrido In-Orden:\n  ";
				inOrden(arbol);
			break;

			case '3':
```

```
                cout << "\tRecorrido Post-Orden:\n  ";
                postOrden(arbol);
            break;
        }
}

//raiz-izquierdo-derecho
void preOrden(Nodo *arbol){
        if (arbol == NULL){
            return;
        }else{
            cout << arbol->info << " - ";
            preOrden(arbol->left);
            preOrden(arbol->right);
        }
}

//izquierdo-derecho-raiz
void inOrden(Nodo *arbol){
        if (arbol == NULL){
            return;
        }else{
            inOrden(arbol->left);
            cout << arbol->info << " - ";
            inOrden(arbol->right);
        }
}

//izquierdo-derecho-raiz
void postOrden(Nodo *arbol){
        if (arbol == NULL){
            return;
        }else{
            postOrden(arbol->left);
            postOrden(arbol->right);
            cout << arbol->info << " - ";
        }
}

void eliminar(Nodo *&arbol, int n){
        if (arbol == NULL){
            return;
        }else{
            if (n < arbol->info){
                eliminar(arbol->left, n);
            }else if (n > arbol->info){
                eliminar(arbol->right, n);
            }else{
                eliminarNodo(arbol);
            }
        }
}

void eliminarNodo(Nodo *&nodoEliminar){
        if (nodoEliminar->left && nodoEliminar->right){
            Nodo *menor = minimo(nodoEliminar->right);
            nodoEliminar->info = menor->info;
            eliminarNodo(menor);
        }else if(nodoEliminar->left){
            reemplazar(nodoEliminar, nodoEliminar->left);
```

```
            destruirNodo(nodoEliminar);
      }else if(nodoEliminar->right){
            reemplazar(nodoEliminar, nodoEliminar->right);
            destruirNodo(nodoEliminar);
      }else{
            Nodo *nulo = NULL;
            reemplazar(nodoEliminar, nulo);
            destruirNodo(nodoEliminar);
      }
}

Nodo *minimo(Nodo *arbol){
      if (arbol == NULL){
            return NULL;
      }else{
            if (arbol->left){
                  return minimo(arbol->left);
            }else{
                  return arbol;
            }
      }
}

void reemplazar(Nodo *&arbol, Nodo *&nuevoNodo){
      if (arbol->daddy){
            if (arbol->info == arbol->daddy->left->info){
                  arbol->daddy->left = nuevoNodo;
            }else if (arbol->info == arbol->daddy->right->info){
                  arbol->daddy->right = nuevoNodo;
            }
      }
      if (nuevoNodo){
            nuevoNodo->daddy = arbol->daddy;
      }
}

void destruirNodo(Nodo *&nodo){
      nodo->left = NULL;
      nodo->right = NULL;

      delete nodo;
}
```

«***delete***» Una expresión con el operador *delete*, primero llama al destructor apropiado (para tipos de clase), y luego llama a una función de desocupación de memoria.