Arboles

Funciones para trabajar con árboles **binarios** de **búsqueda**.

Definiciones: Las listas enlazadas, pilas y colas son estructuras lineales de datos. Un árbol es una estructura de datos bidimensional no lineal. Los nodos de un árbol contienen dos o más enlaces.
Los árboles se utilizan para representar fórmulas algebraicas, para organizar objetos en orden de tal forma que las búsquedas son muy eficientes, y en aplicaciones diversas tales como inteligencia artificial o algoritmos de cifrado. Casi todos los sistemas operativos almacenan sus archivos en árboles o estructuras similares a árboles. Además de las aplicaciones citadas, los árboles se utilizan en diseño de compiladores, proceso de texto y algoritmos de búsqueda.
Intuitivamente el concepto de árbol implica una estructura en la que los datos se organizan de modo que los elementos de información están relacionados entre sí a través de ramas. El árbol genealógico es el ejemplo típico más representativo del concepto de árbol general.
Un árbol consta de un conjunto finito de elementos, denominados nodos y un conjunto finito de líneas dirigidas, denominadas ramas, que conectan los nodos.
Un árbol binario es un árbol en el que ningún nodo puede tener más de dos subárboles. En un árbol binario, cada nodo puede tener, cero, uno o dos hijos (subárboles). Se conoce el nodo de la izquierda como hijo izquierdo y el nodo de la derecha como hijo derecho.
Un árbol binario puede definirse como una estructura recursiva donde cada nodo es el raíz de su propio subárbol y tiene hijos, que son raíces de árboles llamados los subárboles derecho e izquierdo del nodo, respectivamente.
Binarios de búsqueda por el criterio en que se insertan los nodos. Si el valor a insertar es menor al del nodo analizado, entonces se insertan a través del nodo izquierdo, aso contrario se inserta desde el nodo derecho. La forma mas simple de implementación es con funciones recursivas.
Veremos como insertar un nodo en un árbol y como recorrer el árbol. Los recorridos también con funciones recursivas.
Para recorrer un árbol se necesita Visitar los nodos y mostrar la información. Según cuando se muestre la información puede ser PreOrden, si se muestra al principio, InOrden, si se muestra entre ambas visitas, PosOrden si se muestra después de visitar ambos nodos. Si se comienza con el nodo izquierdo la secuencia es la que escribí, si se comienza con el derecho, estos recorridos se llaman inversos. Por eso las alternativas son 6 (no escribiré todas, les dejo algunas a ustedes)

MostrarInFormacion -> VisitarNodoIzquierdo -> VisitarNodoDerecho: PreOrden

VisitarNodoIzquierdo -> MostrarInFormacion -> VisitarNodoDerecho: InOrden

VisitarNodoIzquierdo -> VisitarNodoDerecho -> MostrarInFormacion: PosOrden

MostrarInFormacion -> VisitarNodoDerecho -> VisitarNodoIzquierdo: PreOrdenInverso
VisitarNodoDerecho -> MostrarInFormacion -> VisitarNodoIzquierdo: InOrdenInverso
VisitarNodoDerecho -> VisitarNodoIzquierdo -> MostrarInFormacion: PosOrdenInverso

incluyan las bibliotecas que correspondan

Estructura del nodo es autoreferenciada con un puntero a cada hijo

```
struct NodoArbol {
        nodoArbol *ptrIzq;
        int dato;
        nodoArbol *prtDer;
};
```

Prototipos de funciones

Inserta un nodo en un arbol de búsqueda de enteros, el valor es el entero a insertar ptrArbol es un puntero pasado por referencia

```
void insertaNodo(NodoArbol* &ptrArbol, int valor);
```

Los recorridos son funciones void que reciben en ptrArbol un puntero al nodo rafz, pasado por valor ya que solo muestra el contenido del árbol sin alterar sus valores

```
void inOrden(NodoArbol*  ptrArbol);
void preOrden(NodoArbol*  ptrArbol);
void postOrden(NodoArbol*  ptrArbol);
```

Va un programa de prueba y luego al final el desarrollo de las funciones de modo de conservar la estructura general de las aplicaciones que desarrollamos.

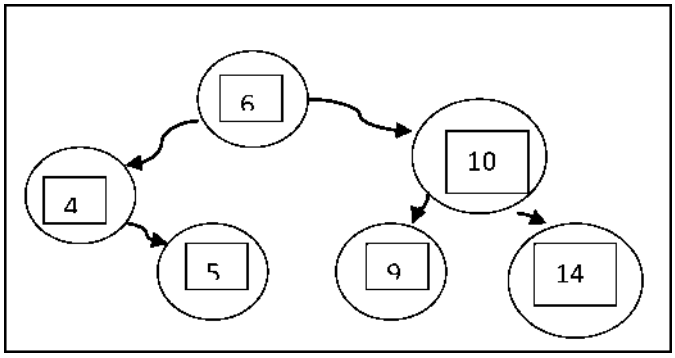
```
int main()
{
```

Partimos de un árbol vacío para luego recorrerlo `NodoArbol* ptrRaiz = NULL;`

El puntero de control ptrRaiz, lo inicializamos en NULL como en toda estructura enlazada

Pruebo insertando valores constantes(6,10,4,5,9,14) de modo que puedan observar como se recorre segun las 6 alternativas (recuerden, 3 de ellas no las hice, HAGAN LAS!!!!!

```
insertaNodo(ptrRaiz,6);
insertaNodo(ptrRaiz,10);
insertaNodo(ptrRaiz,4);
insertaNodo(ptrRaiz,5);
insertaNodo(ptrRaiz,9);
insertaNodo(ptrRaiz,14);
preOrden(ptrRaiz);
```



Estado después de insertar los valores
6 Nodo raiz
4 hijo Izquierdo de 6
10 hijo derecho de 6
5 hijo derecho de 4
9 hijo izquierdo de 10,
14 hijo derecho de 10

La secuencia fue 6, 10, 4, 5, 9,14.

Siempre se comienza por el nodo rafz y se inserta en una hoja, si el valor es mayor se inserta a la derecha y si es menor a la izquierda.

En este caso se inserta primero el valor 6 porque el árbol esta vacío, al ingresar 10 se comienza por el raíz que tiene 6 como 10 es mayor que 6 se va por la rama derecha, y como el puntero derecho en NULL allí se inserta 10. Al ingresar 4, por ser menor a 6 va a la rama izquierda. Al ingresar 5 primero se lo compara con 6, va a la rama izquierda, allí esta 4, como 5 es mayor va a la rama derecha, al ser NULL allí se inserta. Pueden seguir ustedes con toda la secuencia.

Recorridos (pongo solo uno de ejemplo)

```
void insertaNodo(NodoArbol* &ptrArbol, int valor)
{
//la funcion se invoca recursivamente, termina si inserta o si encuentra duplicado
/* si el arbol esta vacío */
 if (ptrArbol == NULL) {
    NodoArbol* aux = new NodoArbol();
    //si encuentra un lugar vacio crea un nodo
    aux->valor = valor;
    //carga el valor en el nodo creado
    aux->ptrIzq = NULL;
     //pone los hijos en NULL
    aux->prtDer = NULL;
    //vincula el nuevo nodo al árbol
    ptrArbol=aux;

} else {
   //Si el dato a insertar es menor que el dato en el nodo actual va por la rama izquierda
   if (valor < ptrArbol->valor)
       {
           insertaNodo(ptrArbol->ptrIzq, valor);
       }
       else
           if (valor > ptrArbol->valor) {
           //Si es mayor va por rama derecha
               insertaNodo(ptrArbol->prtDer, valor);
           } else {
           //si es igual no lo inserta
           cout<<"Valor duplicado";
           }
       }
       return;
   }
```

```
 void preOrden(NodoArbol* ptrArbol){
     if (ptrArbol != NULL) {
         cout<<ptrArbol->valor;
         //muestra la informacion
         preOrden(ptrArbol->ptrIzq);
         //recorre la rama izquierda
         preOrden(ptrArbol->prtDer);
         //recorre la rama derecha
         }
     return;
 }
```