Práctica 18

Arboles Binarios no ordenados

1. Ejecute el programa ***Dev-C++***.
2. En el menú ***Archivo*** seleccione la opción ***Nuevo*** y a continuación ***Archivo Fuente***.
3. Transcriba el código del siguiente programa, mediante el cual se puede crear un árbol binario, que permite ingresar diferentes valores enteros donde nosotros indiquemos, y realizar la búsqueda de un valor determinado mediante las estrategias BEP y BEA.

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

4. A continuación se define la clase ***NodoArbol*** que cuenta con dos apuntadores para los hijos izquierdo y derecho respectivamente y el método constructor que almacena el valor proporcionado y les asigna valores nulos a ambos apuntadores:

```
template<class T>
class NodoArbol {
   public: NodoArbol<T> *HijoIzq;
           NodoArbol<T> *HijoDer;
           NodoArbol(T);
           T Info;
};

template<class T>
NodoArbol<T>::NodoArbol(T v) {
   Info = v;
   HijoIzq=NULL;
   HijoDer=NULL;
}
```

5. Ahora definimos la clase ***Arbol*** en la que existe un nodo raíz que se inicializa con valor Nulo en el método constructor y que cuenta con los métodos de inserción para los hijos de un determinado nodo, ya sea izquierdo o derecho, según se desee y los métodos de búsqueda de un valor en profundidad o en anchura:

```
template<class T>
class Arbol {
   public: NodoArbol<T> *Raiz;
           Arbol();
           NodoArbol<T> *Ins_Izq(T,T);
           NodoArbol<T> *Ins_Der(T,T);
           NodoArbol<T> *BEA(T);
           NodoArbol<T> *BEP(T);
};
```

```
template<class T>
Arbol<T>::Arbol() {
   Raiz=NULL;
}
```

6. Para la inserción de un nodo se indica el valor del nodo padre y se localiza mediante una BEA, excepto en el caso de que no exista el nodo raíz, que tomará el valor dado sin importar el que se haya indicado para el padre. En los demás casos, si encuentra al nodo padre, verifica que no tenga previamente un hijo (ya sea izquierdo o derecho, según el caso) y si no existe procede a crearlo:

```
template<class T>
NodoArbol<T> *Arbol<T>::Ins_Izq(T Padre,T Dato) {
   NodoArbol<T> *Apu;
   if(Raiz!=NULL) {
      Apu=BEA(Padre);
      if(Apu==NULL) return NULL;
      if(Apu->HijoIzq==NULL) return Apu->HijoIzq=new NodoArbol<T>(Dato);
      else return NULL;
   }
   else return Raiz=new NodoArbol<T>(Dato);
}

template<class T>
NodoArbol<T> *Arbol<T>::Ins_Der(T Padre,T Dato) {
   NodoArbol<T> *Apu;
   if(Raiz!=NULL) {
      Apu=BEA(Padre);
      if(Apu==NULL) return NULL;
      if(Apu->HijoDer==NULL) return Apu->HijoDer=new NodoArbol<T>(Dato);
      else return NULL;
   }
   else return Raiz=new NodoArbol<T>(Dato);
}
```

7. En la búsqueda en profundidad se crea un arreglo de apuntadores que se usará como pila para almacenar los nodos que faltan de visitar; para cada nodo visitado, se insertará en la pila primero el hijo derecho y luego el izquierdo si es que existen, luego se visitará al que se encuentre en la cima de la pila hasta encontrar el valor o vaciar la pila (condición Cima==-1).
8. Se utiliza un arreglo pequeño porque se necesitaría un árbol muy alto para llenar la pila:

```
template<class T>
NodoArbol<T> *Arbol<T>::BEP(T Dato) {
   NodoArbol<T> *Apu=Raiz;
   if (Raiz != NULL) {
      int Cima=-1;
      NodoArbol<T> *pila[10];
```

```
    cout<<endl<<"inicio la busqueda del dato: "<<Dato<<endl;
    do {
      cout<<endl<<"checando el nodo: "<<Apu->Info<<endl;
      if (Apu->Info==Dato) {
        cout<<"Nodo encontrado."<<endl;
        return Apu;
      }
      else {
        if (Apu->HijoDer !=NULL) {
          Cima++;
          pila[Cima]=Apu->HijoDer;
          if (Cima>=10) cout <<"OJO: Pila llena";
        }
        if (Apu->HijoIzq !=NULL) {
          Cima++;
          pila[Cima]=Apu->HijoIzq;
          if (Cima>=10) cout <<"OJO: Pila llena";
        }
        cout << "estado actual de la pila: ";
        for (int i=0;i<=Cima;i++) cout <<setw(6)<<pila[i]->Info;
        if(Cima==-1) break;
        Apu=pila[Cima];
        Cima--;
      }
    } while (true);
  }
  cout<<endl;
  return NULL;
}
```

9. En la búsqueda por anchura se utiliza un arreglo de apuntadores para simular una cola circular, para cada nodo visitado se inserta al final de la cola el hijo izquierdo y luego el derecho, si es que existen, luego se extrae el nodo que esté al inicio de la cola.
10. Este arreglo cuenta con 10 elementos y se requeriría un árbol muy denso para llenar la cola:

```
template<class T>
NodoArbol<T> *Arbol<T>::BEA(T Dato) {
  NodoArbol<T> *Apu=Raiz;
  if (Raiz != NULL) {
    int i,Frente=-1,Final=0;
    NodoArbol<T> *cola[10];
    cout<<endl<<"inicio la busqueda del dato: "<<Dato<<endl;
    do {
      cout<<endl<<"checando el nodo: "<<Apu->Info<<endl;
      if (Apu->Info==Dato) {
        cout<<"Nodo encontrado."<<endl;
        return Apu;
```

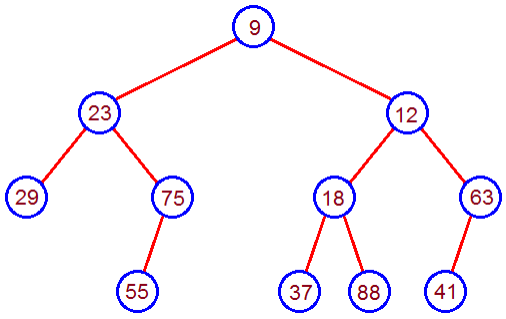
```
            }
            else {
               if (Apu->HijoIzq !=NULL) {
                  cola[Final]=Apu->HijoIzq;
                  Final++;
                  if (Final>10) Final=0;
               }
               if (Apu->HijoDer !=NULL) {
                  cola[Final]=Apu->HijoDer;
                  Final++;
                  if (Final>10) Final=0;
               }
               if (Final==Frente+1) break;
               cout << "estado actual de la cola: ";
               i=Frente+1;
               while (i<Final){
                  cout <<setw(6)<<cola[i]->Info;
                  i++;
                  if (i>10) i=0;
               }
               Frente++;
               Apu=cola[Frente];
               if (Frente>=10) Frente=-1;
            }
         } while (Final!=Frente);
      }
      cout<<endl;
      return NULL;
   }
```

11. Mediante el programa principal elegimos las acciones a realizar:

```
   int main (void) {
      Arbol<int> arbol;
      NodoArbol<int> *Apu;
      char Opc;
      int Valor,Padre;
      do {
         cout<<endl<<"Implementacion de un arbol Binario\n";
         cout<<"\n1.- Insertar un nodo izquierdo\n2.- Insertar un nodo derecho\n";
         cout<<"3.- Realizar busqueda en Profundidad\n4.- Realizar busqueda en Anchura\n";
         cout<<"5.- Salir\n\nOpcion: ";
         cin>>Opc;
         switch(Opc) {
            case '1': cout<<endl<<"Introduce el valor del Nodo Padre: ";
                      cin>>Padre;
                      cout<<endl<<"Introduce el valor a insertar: ";
                      cin>>Valor;
```

```
                Apu=arbol.Ins_Izq(Padre,Valor);
                if (Apu==NULL) cout<<"No procedio la insercion"<<endl;
                else cout <<"Se inserto el valor: "<<Apu->Info<<endl;
                break;
        case '2': cout<<endl<<"Introduce el valor del Nodo Padre: ";
                cin>>Padre;
                cout<<endl<<"Introduce el valor a insertar: ";
                cin>>Valor;
                Apu=arbol.Ins_Der(Padre,Valor);
                if (Apu==NULL) cout<<"No procedio la insercion"<<endl;
                else cout <<"Se inserto el valor: "<<Apu->Info<<endl;
                break;
        case '3': cout<<endl<<"Introduce un valor a buscar: ";
                cin>>Valor;
                Apu=arbol.BEP(Valor);
                if (Apu==NULL) cout<<"No se encontro el valor"<<endl;
                break;
        case '4': cout<<endl<<"Introduce un valor a buscar: ";
                cin>>Valor;
                Apu=arbol.BEA(Valor);
                if (Apu==NULL) cout<<"No se encontro el valor"<<endl;
                break;
      }
    } while(Opc!='5');
}
```

12. Ahora localice el icono de ***Compilar*** que está al inicio de la barra de herramientas y oprímalo, proporcione el nombre de ***ejercicio 18*** y elija el destino que guste para guardarlo.
13. Ejecute su programa e ingrese el árbol de la figura, siguiendo la secuencia que guste, obviamente comenzando con el nodo 9 en la raíz:



14. Pruebe insertar un hijo en de un valor no presente, así como un hijo de un nodo que ya tenga otro hijo previamente.
15. Una vez hecha la inserción, realice la búsqueda en anchura y en profundidad de diversos valores, tanto existentes en el árbol como inexistentes.
16. Fin de la Práctica.