```
#include <iostream>
#define MAX 1000
struct ArbolBinario{
    int dato;
    struct ArbolBinario* iz;
    struct ArbolBinario* de;
    struct ArbolBinario* padre;
};
typedef struct ArbolBinario Narbol;
struct pila{
    Narbol* entradas[MAX];
    int tamanio;
    int tope;
};
typedef struct pila NpilaE;
struct pilaD{
    Narbol* NODOS;
    struct pilaD* link;
};
typedef struct pilaD Npila;
struct cola{
    Narbol* NODO;
    struct cola* link;
};
typedef struct cola Ncola;
struct punteros_cola{
    Ncola* frente;
    Ncola* fondo;
};
typedef struct punteros_cola Npunt;
//FUMCIONES DE PILA ESTATICA
```

```
void Alta_pila(NpilaE*& pila, Narbol* raiz);
Narbol* Baja_pila(NpilaE*& pila);
bool pila_vacia(NpilaE* pila);
//FUNCIONES DE PILA DINAMICA
void Alta_pilaDIN(Npila*& pila, Narbol* raiz);
Narbol* Baja_pilaDIN(Npila*& pila);
bool pila_vaciaDIN(Npila* pila);
//FUNCIONES DE cola
void Alta_cola(Npunt &punteros, Narbol* nodo);
Narbol* Baja_cola(Npunt &punteros);
bool Vacia_cola(Npunt &punteros);
//FUNCIONES DE ARBOL
void addOrdit(Narbol* &raiz, int dat);
void addOrdRe(Narbol* &raiz, int dat);
Narbol* new_nodo(int dat);
//BARRIDOS RECURSIVOS
void preOrdenRecur(Narbol*raiz);
void posOrdenRecur(Narbol*raiz);
void InOrdenRecur(Narbol*raiz);
void drawre(Narbol*raiz);
//BARRIDOS ITERATIVOS
void Barrido_por_nivel(Narbol* raiz);
void preOrdenIT(Narbol* raiz);
void posOrdenIT(Narbol* raiz);
void inOrdenIT(Narbol* raiz);
//LLENO INCOMPLETO DESCENDENCIA
int BuscarDescendencia(Narbol* raiz, int dat);
bool arbolComp(Narbol* raiz);
bool arbolLleno(Narbol* raiz);
bool nodo_hoja(Narbol* raiz);
//CALCULAR NIVEL Y ALTURA
```

```
int calcular_nivel_iterativa(Narbol* raiz);
int calcularAlturaIterativa(Narbol* raiz);
int abinario_altura_recursivo(Narbol *arbol);
int main(){
    Narbol* Nuevo = new Narbol;
    Nuevo = NULL;

}
//FUNCIONES DE ARBOL
Narbol* new_nodo(int dat){
    Narbol* nuevo_nodo = new Narbol;
    nuevo_nodo->dato = dat;
    nuevo_nodo->iz = NULL;
    nuevo_nodo->de = NULL;
    return nuevo_nodo;
}
void addOrdit(Narbol* &raiz, int dat){
    Narbol* auxiliar = raiz;
    Narbol* control = raiz;
    Narbol* nuevo_nodo = new_nodo(dat);
    if(raiz==NULL){
        raiz = nuevo_nodo;
        nuevo_nodo->padre = NULL;
    }else{
        while(auxiliar==control){
            if(auxiliar->dato>dat){
                if(auxiliar->iz==NULL){
                    auxiliar->iz=nuevo_nodo;
                    nuevo_nodo->padre = auxiliar;
                    control = NULL;
                }else{
```

```
                    auxiliar = auxiliar->iz;
                    control = auxiliar;
                }
            }else{
                if(auxiliar->de==NULL){
                    auxiliar->de=nuevo_nodo;
                    nuevo_nodo->padre = auxiliar;
                    control = NULL;
                }else{
                    auxiliar = auxiliar->de;
                    control = auxiliar;
                }
            }
        }
    }
}
void addOrdRe(Narbol* &raiz, int dat){
    Narbol* auxiliar = raiz;
    if(raiz==NULL){
        raiz = new_nodo(dat);
        raiz->padre = NULL;
    }else{
        if(auxiliar->dato>dat){
            if(auxiliar->iz==NULL){
                auxiliar->iz=new_nodo(dat);
                auxiliar->iz->padre = auxiliar;
            }else{
               addOrdRe(auxiliar->iz, dat);
            }
        }else{
            if(auxiliar->de==NULL){
```

```
                auxiliar->de=new_nodo(dat);
                auxiliar->de->padre = auxiliar;
            }else{
                addOrdRe(auxiliar->iz, dat);
            }
        }
    }
}
//FUNCIONES LLENO INCOMPLETO DESCENDENCIA
bool arbolComp(Narbol* raiz){
    bool bandera = true;
    if(raiz!=NULL){
        Npunt cola;
        cola.fondo = NULL;
        cola.frente = NULL;
        Narbol* aux;
        Alta_cola(cola, raiz);
        while(!Vacia_cola(cola)&&bandera){
            aux = Baja_cola(cola);
            if(aux->iz!=NULL&&aux->de==NULL||aux->iz==NULL&&aux->de!=NULL){
                bandera = false;
            }else{
                if(aux->iz!=NULL) Alta_cola(cola, aux->iz);
                if(aux->de!=NULL) Alta_cola(cola, aux->de);
            }
        }
        while (!Vacia_cola(cola)){
            aux = Baja_cola(cola);
        }
    }
    return bandera;
```

```
}
bool arbolLleno(Narbol* raiz){
    int cont = 0;
    int x = calcularAlturaIterativa(raiz);
    int aux = 1;
    NpilaE* nuevaPIla = new NpilaE;
    nuevaPIla->tamanio = 0;
    nuevaPIla->tope = MAX;
    Alta_pila(nuevaPIla, raiz);
    while(!pila_vacia(nuevaPIla)){
        raiz = Baja_pila(nuevaPIla);
        if(raiz->iz!=NULL) Alta_pila(nuevaPIla,raiz->iz);
        if(raiz->de!=NULL) Alta_pila(nuevaPIla, raiz->de);
        cont++;
    }
    for(int i=0; i<x; i++){
        aux*=2;
    }
    return aux-1==cont;
}
int BuscarDescendencia(Narbol* raiz, int dat){
    int cont = -1;
    bool band = true;
    NpilaE* nuevaPIla = new NpilaE;
    nuevaPIla->tamanio = 0;
    nuevaPIla->tope = MAX;
    if(raiz==NULL){
        std::cout<<"Pila vacia"<<std::endl;
        return 0;
    }else{
        while(band){
```

```
            if(raiz->dato!=dat){
                if(raiz->dato<dat){
                    if(raiz->de!=NULL) raiz = raiz->de;
                    else{
                        std::cout<<"Dato no encontrado en el
arbol"<<std::endl;
                        band = false;
                    }
                }else{
                    if(raiz->iz!=NULL) raiz = raiz->iz;
                    else{
                        std::cout<<"Dato no encontrado en el
arbol"<<std::endl;
                        band = false;
                    }
                }
            }else{
                Alta_pila(nuevaPIla, raiz);
                while(!pila_vacia(nuevaPIla)){
                    raiz = Baja_pila(nuevaPIla);
                    if(raiz->iz!=NULL) Alta_pila(nuevaPIla,raiz->iz);
                    if(raiz->de!=NULL) Alta_pila(nuevaPIla, raiz->de);
                    cont++;
                }
                band = false;
            }
        }
    }
    return cont;
}
bool nodo_hoja(Narbol* raiz){
    return (raiz->iz==NULL&&raiz->de==NULL);
```

```
}
//FUNCIONES DE ALTURA Y NIVEL
int calcular_nivel_iterativa(Narbol* raiz){
    return calcularAlturaIterativa(raiz)-1;
}
int calcularAlturaIterativa(Narbol* raiz){
    int altura = 0;
    if (raiz == NULL) return altura;
        Narbol* aux = raiz;
        Npunt cola;
        int tamanioCola = 0;
        int cantNodos;
        cola.fondo = NULL;
        cola.frente = NULL;
        Alta_cola(cola, raiz);
        tamanioCola++;
        while (!Vacia_cola(cola)){
            altura++;
            cantNodos = tamanioCola;
            while (cantNodos > 0){
                aux = Baja_cola(cola);
                tamanioCola--;
                cantNodos--;
                if (aux->iz != NULL) {
                    Alta_cola(cola, aux->iz);
                    tamanioCola++;
                }
                if (aux->de != NULL) {
                    Alta_cola(cola, aux->de);
                    tamanioCola++;
                }
```

```
            }
        }
    return altura;
}
int abinario_altura_recursivo(Narbol *arbol){
    int alt_iz,alt_de;
    if(arbol==NULL)return 1;
    alt_iz=abinario_altura_recursivo(arbol->iz);
    alt_de=abinario_altura_recursivo(arbol->de);
    if(alt_iz<alt_de)return alt_iz +1;
    else return alt_de+1;
}
//FUNCIONES BARRIDOS RECURSIVOS
void posOrdenRecur(Narbol* raiz){
    if(raiz==NULL) return;
    posOrdenRecur(raiz->iz);
    posOrdenRecur(raiz->de);
    std::cout<<"Dato: "<<raiz->dato<<std::endl;
}
void preOrdenRecur(Narbol* raiz){
    if(raiz==NULL) return;
    std::cout<<"Dato: "<<raiz->dato<<std::endl;
    preOrdenRecur(raiz->iz);
    preOrdenRecur(raiz->de);
}
void InOrdenRecur(Narbol* raiz){
    if(raiz==NULL) return;
    InOrdenRecur(raiz->iz);
    std::cout<<"Dato: "<<raiz->dato<<std::endl;
    InOrdenRecur(raiz->de);
}
```

```
//FUNCIONES DE BARRIDOS ITERATIVOS
void Barrido_por_nivel(Narbol* raiz){
    if(raiz == NULL){
        std::cout<<"arbol vacio"<<std::endl;
    }else{
        Narbol* aux;
        Npunt puntero;
        puntero.frente=NULL;
        puntero.fondo=NULL;
        Alta_cola(puntero, raiz);
        while(!Vacia_cola(puntero)){
            aux = Baja_cola(puntero);
            if(aux->iz!=NULL)Alta_cola(puntero, aux->iz);
            if(aux->de!=NULL)Alta_cola(puntero, aux->de);
            std::cout<<aux->dato<<"->";
        }
        std::cout<<std::endl;
    }
}
void preOrdenIT(Narbol* raiz){
    if(raiz==NULL){
        std::cout<<"ARBOL VACIO"<<std::endl;
    }else{
        Narbol* aux = raiz;
        Npila* nuevapila = NULL;
        Alta_pilaDIN(nuevapila, aux);
        if(aux!=NULL){
            while(!pila_vaciaDIN(nuevapila)){
                aux = Baja_pilaDIN(nuevapila);
                std::cout<<aux->dato<<" -> ";
                if(aux->de!=NULL) Alta_pilaDIN(nuevapila, aux->de);
```

```
                if(aux->iz!=NULL) Alta_pilaDIN(nuevapila, aux->iz);
            }
        }
    }
}
void posOrdenIT(Narbol* raiz){
    if(raiz==NULL){
        std::cout<<"ARBOL VACIO"<<std::endl;
    }else{
        Narbol* aux = raiz;
        Npila* nuevapila = NULL;
        Npila* pilaAux = NULL;
        Alta_pilaDIN(nuevapila, aux);
        if(aux!=NULL){
            while(!pila_vaciaDIN(nuevapila)){
                aux = Baja_pilaDIN(nuevapila);
                Alta_pilaDIN(pilaAux, aux);
                if(aux->iz!=NULL) Alta_pilaDIN(nuevapila, aux->iz);
                if(aux->de!=NULL) Alta_pilaDIN(nuevapila, aux->de);
            }
        }
        while(!pila_vaciaDIN(pilaAux)){
            Narbol* aux = Baja_pilaDIN(pilaAux);
            std::cout<<aux->dato<<" -> ";
        }
    }
}
void inOrdenIT(Narbol* raiz){
    if(raiz==NULL){
        std::cout<<"ARBOL VACIO"<<std::endl;
    }else{
```

```
        bool band = true;
        Narbol* aux = raiz;
        Npila* nuevapila = NULL;
        while(band){
            if(aux!=NULL){
                Alta_pilaDIN(nuevapila, aux);
                aux = aux->iz;
            }else{
                if(!pila_vaciaDIN(nuevapila)){
                    aux = Baja_pilaDIN(nuevapila);
                    std::cout<<aux->dato<<" -> ";
                    aux = aux->de;
                }else{
                    band = false;
                }
            }
        }
    }
}
//FUNCIONES PILA ESTATICA
void Alta_pila(NpilaE*& pila, Narbol* raiz){
    if(pila->tamanio<pila->tope){
        pila->entradas[pila->tamanio] = raiz;
        pila->tamanio++;
    }else return;
}
Narbol* Baja_pila(NpilaE*& pila){
    Narbol* aux;
    if(!pila_vacia(pila)){
        pila->tamanio--;
        aux = pila->entradas[pila->tamanio];
```

```
    }
    return aux;
}
bool pila_vacia(NpilaE* pila){
    return pila->tamanio == 0;
}
//FUNCIONES PILA DINAMICA

void Alta_pilaDIN(Npila*& pila, Narbol* raiz){
    Npila* nuevo_nodo = new Npila;
    nuevo_nodo->NODOS = raiz;
    nuevo_nodo->link = pila;
    pila = nuevo_nodo;
}
Narbol* Baja_pilaDIN(Npila*& pila){
    Narbol* nodo = pila->NODOS;
    Npila* aux = pila;
    pila = pila->link;
    delete aux;
    return nodo;
}
bool pila_vaciaDIN(Npila* pila){
    return pila==NULL;
}   //CODIGO DE COLA//
void Alta_cola(Npunt &punteros, Narbol* nodo){
    Ncola* nNodo = new Ncola;
    nNodo->NODO = nodo;
    nNodo->link = NULL;
    if(punteros.frente==NULL){
        punteros.fondo = nNodo;
        punteros.frente = punteros.fondo;
```

```
    }else{
        punteros.fondo->link = nNodo;
        punteros.fondo = nNodo;
    }
}
Narbol* Baja_cola(Npunt &punteros){
    Ncola* nNodo = punteros.frente;
    Narbol* nodo;
    if(punteros.frente!=NULL){
        nodo = nNodo->NODO;
    }
    punteros.frente=punteros.frente->link;
    if(punteros.frente==NULL){
        punteros.fondo = NULL;
    }
    delete nNodo;
    return nodo;
}
bool Vacia_cola(Npunt &punteros){
    return (punteros.frente==NULL&&punteros.fondo==NULL);
}
```