UNIVERSIDAD NEBRIJA

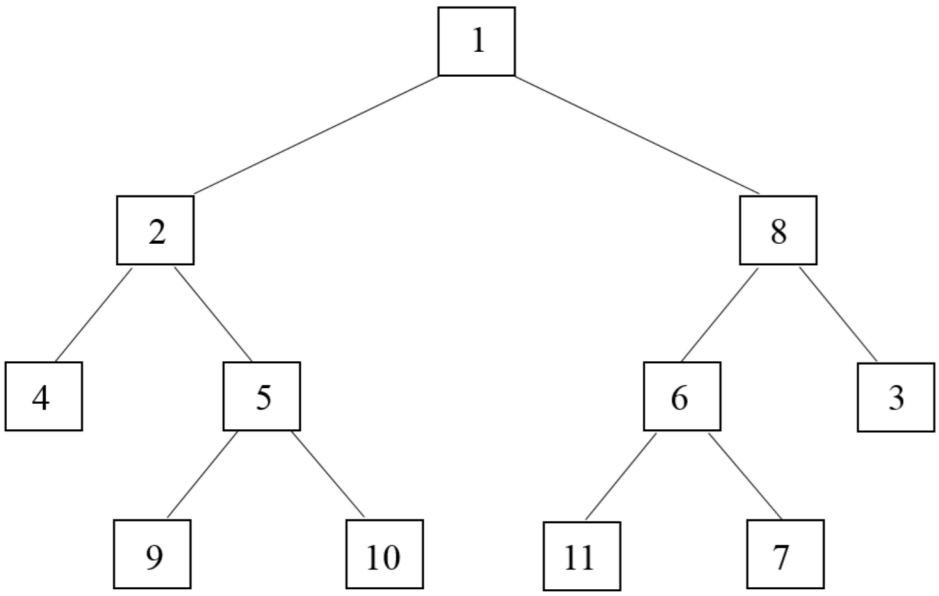
Escuela Politécnica Superior

IIN151 – Estructuras de datos y algoritmos

Prácticas de laboratorio 3

Nombre: _____

Un árbol es un tipo abstracto de datos no lineal. El TAD árbol define una colección de elementos denominados nodos, donde uno de los nodos del árbol es el nodo raíz e impone una jerarquía entre los nodos. Los nodos de un árbol, como los elementos de una lista, tienen un tipo de dato determinado.



El TAD árbol binario es una colección de nodos donde cada uno de ellos tiene como máximo dos descendientes, el hijo izquierdo y el hijo derecho.

La interfaz del TAD árbol binario.

inicializa(R) Inicializa R a un árbol vacío

  Precondición: Ninguna
  Postcondición: Un árbol binario vacío

vacio(R) Devuelve verdadero si el árbol R está vacío y falso en cualquier otro caso
  Precondición: Ninguna
  Postcondición: **true** si el árbol R está vacío y **false** e.o.c.

insertaNodo(v, izquierdo, derecho) Crea un nuevo nodo con el valor v e hijos izquierdo y derecho

  Precondición: Ninguna
  Postcondición: La posición del nuevo nodo

insertaRaiz(R, v, izquierdo, derecho) Crea el nodo raíz de R con el valor v e hijos izquierdo y derecho

  Precondición: Ninguna
  Postcondición: Un árbol con raíz en el nodo R

insertaHijoIzquierdo(R, r) El hijo izquierdo de R toma el valor del subárbol r

  Precondición: Ninguna
  Postcondición: El hijo izquierdo de R toma el valor de r

insertaHijoDerecho(R, r) El hijo derecho de R toma el valor del subárbol r
  Precondición: Ninguna
  Postcondición: El hijo derecho de R toma el valor de r

busca(R, v) Busca el valor v en el árbol R
  Precondición: Ninguna
  Postcondición: Devuelve el primer nodo de R que almacena
                 el valor v

preorder(R) Recorre los nodos del árbol R en “preorder”

  Precondición: Ninguna
  Postcondición: Ninguna

inorder(R) Recorre los nodos del árbol R en “inorder”
  Precondición: Ninguna
  Postcondición: Ninguna

```
postorder(R) Recorre los nodos del árbol R en “postorder”
  Precondición:  Ninguna
  Postcondición: Ninguna
```

Implemente el TAD árbol binario utilizando el siguiente código.

```
struct NodoArbol {
   char dato;
   NodoArbol *izquierdo, *derecho;
};

struct TArbolBinario {
   NodoArbol *raiz;
};

void inicializa(TArbolBinario *R) {
   R->raiz = NULL;
}

bool vacio(TArbolBinario *R) {
   return (R->raiz == NULL);
}

NodoArbol* insertaNodo(char v, NodoArbol *izquierdo,
                       NodoArbol *derecho) {
   NodoArbol *nodo = new NodoArbol;
   nodo->dato = v;
   nodo->izquierdo = izquierdo;
   nodo->derecho = derecho;

   return nodo;
}

void insertaRaiz(NodoArbol *&R, char v, NodoArbol *izquierdo,
                 NodoArbol *derecho) {
   R = insertaNodo(v, izquierdo, derecho);
}

void insertaHijoIzquierdo(NodoArbol *&R, NodoArbol *izquierdo) {
   R->izquierdo = izquierdo;
}

void insertaHijoDerecho(NodoArbol *&R, NodoArbol *derecho) {
   R->derecho = derecho;
}
```

```
NodoArbol* busca(NodoArbol *R, char v) {

   // recorre en profundidad los nodos del árbol R,
   // devuelve el primer nodo que almacena el valor v

}

int altura(NodoArbol *R) {

   // si R es un árbol vacío, su altura es -1
   // si R no es un árbol vacío, su altura es 1 más la
   // altura mayor de sus ramas izquierda y derecha

}

int nodos(NodoArbol *R) {

   // cuenta el número de nodos del árbol R

}

void preorder(NodoArbol *R) {
   if (R != NULL) {
      std::cout << R->dato << " ";
      preorder(R->izquierdo);
      preorder(R->derecho);
   }
}

void inorder(NodoArbol *R) {
   if (R != NULL) {
      inorder(R->izquierdo);
      std::cout << R->dato << " ";
      inorder(R->derecho);
   }
}

void postorder(NodoArbol *R) {
   if (R != NULL) {
      postorder(R->izquierdo);
      postorder(R->derecho);
      std::cout << R->dato << " ";
   }
}
```

```
void imprimeArbol(struct TArbolBinario *R) {
   std::cout << "\n";
   std::cout << "Preorder  ";
   preorder(R->raiz);
   std::cout << "\n";
   std::cout << "Inorder   ";
   inorder(R->raiz);
   std::cout << "\n";
   std::cout << "Postorder ";
   postorder(R->raiz);
   std::cout << "\n";
}
```

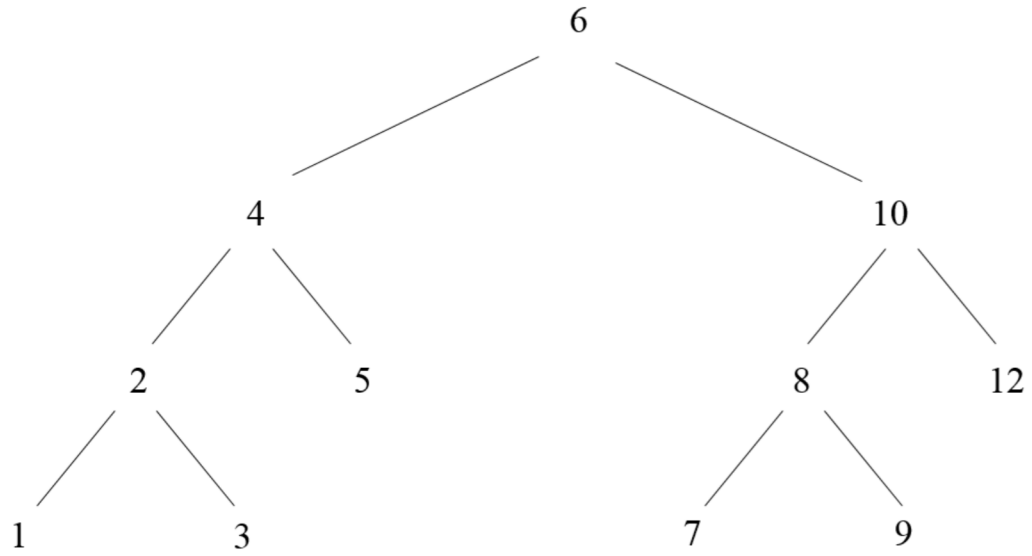
El programa de prueba del TAD árbol binario.

```
int main() {
   struct TArbolBinario *R = new TArbolBinario;
   struct NodoArbol *raiz, *izquierdo, *derecho;

   // arbol construido de las hojas a la raíz

   inicializa(R);

   izquierdo = insertaNodo('*', insertaNodo('3', NULL, NULL),
               insertaNodo('2', NULL, NULL));
   derecho = insertaNodo('-', insertaNodo('5', NULL, NULL),
             insertaNodo('3', NULL, NULL));
   insertaRaiz(R->raiz, '+', izquierdo, derecho);

   imprimeArbol(R);


   std::cout << "\n";
   std::cout << "Altura del arbol " << altura(R->raiz) << " con " <<
             nodos(R->raiz) << " nodos \n";

   if (busca(R->raiz, '4') != NULL)
      std::cout << "El nodo 4 forma parte del arbol R " << "\n";
   else
      std::cout << "El nodo 4 no forma parte del arbol R " << "\n";

   if (busca(R->raiz, '2') != NULL)
      std::cout << "El nodo 2 forma parte del arbol R " << "\n";
   else
      std::cout << "El nodo 2 no forma parte del arbol R " << "\n";
```

```
    // arbol construido de la raíz a las hojas

    struct TArbolBinario *T = new TArbolBinario;

    inicializa(T);

    insertaRaiz(T->raiz, '*', NULL, NULL);
    izquierdo = insertaNodo('*', insertaNodo('4', NULL, NULL),
                insertaNodo('5', NULL, NULL));
    derecho = insertaNodo('+', insertaNodo('6', NULL, NULL),
              insertaNodo('7', NULL, NULL));
    insertaHijoIzquierdo(T->raiz, izquierdo);
    insertaHijoDerecho(T->raiz, derecho);

    imprimeArbol(T);

    std::cout << "\n";
    std::cout << "Altura del arbol " << altura(T->raiz) << " con " <<
              nodos(T->raiz) << " nodos \n";

    if (busca(T->raiz, '4') != NULL)
       std::cout << "El nodo 4 forma parte del arbol T " << "\n";
    else
       std::cout << "El nodo 4 no forma parte del arbol T " << "\n";

    if (busca(T->raiz, '8') != NULL)
       std::cout << "El nodo 8 forma parte del arbol T " << "\n";
    else
       std::cout << "El nodo 8 no forma parte del arbol T " << "\n";

    return 0;
}
```

Un árbol binario de búsqueda (Binary Search Tree) es un árbol binario en el que para cualquier nodo se cumplen las siguientes condiciones:

- El hijo izquierdo, si no es nulo, almacena un valor menor que el del nodo padre
- El hijo derecho, si no es nulo, almacena un valor mayor que el del nodo padre



La interfaz del TAD árbol binario de búsqueda.

```
inicializa(R) Inicializa R a un árbol vacío

  Precondición:  Ninguna
  Postcondición: Un árbol binario de búsqueda vacío

vacio(R) Devuelve verdadero si el árbol R está vacío y falso en
cualquier otro caso

  Precondición:  Ninguna
  Postcondición: true si el árbol R está vacío y false e.o.c.

inserta(R, v) Inserta un nodo con el valor v

  Precondición:  Ninguna
  Postcondición: El nodo con el valor v ocupa la posición que
                 le corresponde en el árbol R
```

```
busca(R, v) Busca el valor v en el árbol R

  Precondición:  Ninguna
  Postcondición: true si v está en el árbol R y false e.o.c.

elimina(R, v) Elimina el nodo de R que almacena el valor v

  Precondición:  Ninguna
  Postcondición: El árbol R ya no tiene un nodo con el valor v

preorder(R) Recorre los nodos del árbol R en “preorder”

  Precondición:  Ninguna
  Postcondición: Ninguna

inorder(R) Recorre los nodos del árbol R en “inorder”
  Precondición:  Ninguna
  Postcondición: Ninguna


postorder(R) Recorre los nodos del árbol R en “postorder”
  Precondición:  Ninguna
  Postcondición: Ninguna
```

Implemente el TAD árbol binario de búsqueda utilizando el siguiente código.

```
struct NodoArbol {
   int dato;
   NodoArbol *izquierdo, *derecho;
};

struct TArbolBinarioBusqueda {
   NodoArbol *raiz;
};

void inicializa(TArbolBinarioBusqueda *R) {
   R->raiz = NULL;
}

bool vacio(TArbolBinarioBusqueda *R) {
   return (R->raiz == NULL);
}
```

```
NodoArbol* insertaNodo(int v) {
   NodoArbol *nodo = new NodoArbol;
   nodo->dato = v;
   nodo->izquierdo = NULL;
   nodo->derecho = NULL;

   return nodo;
}

bool inserta(NodoArbol *&R, int v) {

   // inserta un nuevo nodo con el valor v en el árbol R
   // de forma que R sea un árbol binario de búsqueda

}

NodoArbol* busca(NodoArbol *R, int v) {

   // si el árbol R contiene un nodo con el valor v, devuelve
   // el puntero a ese nodo, e.o.c. devuelve NULL

}

NodoArbol* minimo(NodoArbol *R) {

   // devuelve el puntero al nodo con el valor mínimo de
   // un árbol binario de búsqueda

}

NodoArbol* maximo(NodoArbol *R)

   // devuelve el puntero al nodo con el valor máximo de
   // un árbol binario de búsqueda

}

int eliminaMinimo(NodoArbol *&R) {

   // elimina el valor mínimo del árbol R

}

void elimina(NodoArbol *&R, int v) {

   // elimina el valor v del árbol R, de manera que R
   // sea un árbol binario de búsqueda

}
```

```
int altura(NodoArbol *R) {

   // si R es un árbol vacío, su altura es -1
   // si R no es un árbol vacío, su altura es 1 más la
   // altura mayor de sus ramas izquierda y derecha

}

int nodos(NodoArbol *R) {

   // cuenta el número de nodos del árbol R

}

void preorder(NodoArbol *R) {
   if (R != NULL) {
      std::cout << R->dato << " ";
      preorder(R->izquierdo);
      preorder(R->derecho);
   }
}

void inorder(NodoArbol *R) {
   if (R != NULL) {
      inorder(R->izquierdo);
      std::cout << R->dato << " ";
      inorder(R->derecho);
   }
}

void postorder(NodoArbol *R) {
   if (R != NULL) {
      postorder(R->izquierdo);
      postorder(R->derecho);
      std::cout << R->dato << " ";
   }
}

void imprimeArbol(struct TArbolBinario *R) {
   std::cout << "\n";
   std::cout << "Preorder  ";
   preorder(R->raiz);
   std::cout << "\n";
   std::cout << "Inorder   ";
   inorder(R->raiz);
   std::cout << "\n";
   std::cout << "Postorder ";
   postorder(R->raiz);
   std::cout << "\n";
}
```

El programa de prueba del TAD árbol binario de búsqueda.

```
int main() {
   struct TArbolBinarioBusqueda *R = new TArbolBinarioBusqueda;

   inicializa(R);

   inserta(R->raiz, 8);
   inserta(R->raiz, 6);
   inserta(R->raiz, 4);
   inserta(R->raiz, 7);
   inserta(R->raiz, 5);
   inserta(R->raiz, 10);
   inserta(R->raiz, 9);
   inserta(R->raiz, 15);

   imprimeArbol(R);

   std::cout << "\nelimina min: " << eliminaMinimo(R->raiz) << "\n";

   imprimeArbol(R);

   //std::cout << "min: " << min(R->raiz)->dato << " minimo: " <<
               minimo(R->raiz)->dato << "\n";
   //std::cout << "max: " << max(R->raiz)->dato << " maximo: " <<
               maximo(R->raiz)->dato << "\n";
   //std::cout << "busca 10 " << busca(R->raiz, 10)->dato << "\n";

   // arbol binario de búsqueda, inorder 5, 7, 10, 12, 14, 15, 18

   struct TArbolBinarioBusqueda *T = new TArbolBinarioBusqueda;

   inicializa(T);

   inserta(T->raiz, 10);
   inserta(T->raiz, 5);
   inserta(T->raiz, 14);
   inserta(T->raiz, 7);
   inserta(T->raiz, 18);
   inserta(T->raiz, 12);
   inserta(T->raiz, 15);

   imprimeArbol(T);
   std::cout << "\nelimina 10 \n";
   elimina(T->raiz, 10);
   imprimeArbol(T);

   return 0;
}
```