

# Programa 02: Árbol ABB

10 de octubre de 2022

Carlos Reyes Rico. 217458353.

Estructura De Datos 2.



## OBJETIVO

Codificar un ABB con las siguientes características:

- 1) Debe de hacer uso de memoria dinámica (apuntadores).
- 2) Crear una clase interfaz o en el main para el uso completo de todas sus operaciones
- 3) Crear los métodos de guardar/cargar para toda el ABB con la estrategia de delimitadores, el archivo debe llamarse file01.txt
- 4) Crear una clase que será su dato en cada nodo. (3 atributos privados, sus métodos get y set, así como los métodos que crean que sean de utilidad) Las operaciones de ABB deben estar completa, incluyendo la de eliminar.
- 5) Las operaciones de ABB deben estar completa, incluyendo la de eliminar.

**Nota:** Lean y vayan programando, la siguiente clase establecemos fecha y resolvemos dudas puntuales del programa.

a) A la plataforma debe de subir su programa y un reporte con impresiones de pantalla del funcionamiento de su programa.

a.1) Debe de tener una explicación breve que hace su programa

a.2) Debe de explicar cómo organizo sus datos en la TDA

a.3) Debe de explicar la estrategia que uso para almacenar y recuperar su dato.

a.4) El reporte debe de tener el siguiente nombre reporte01-PrimerApellido SegundoApellido Nombre.pdf

b) El nombre del programa debe ser prog01-PrimerApellido SegundoApellido Nombre.cpp o comprimir todo el proyecto en un solo archivo esté nombre

# METODOS

## 1. Insertar.

El método insertar se encarga de agregar y acomodar los datos dentro de nuestro ABB, siguiendo la regla básica de los ABB.

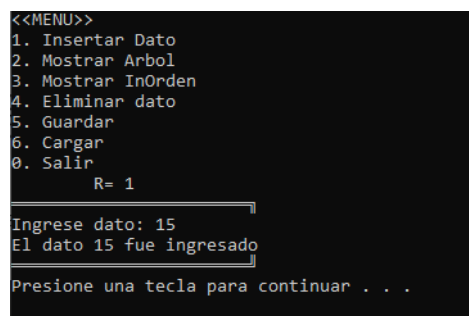
Para la inserción de datos se respalda del metodo crearNodo el cual se encarga de crear el nodo a agregar.

- Código

```
nodo *abb::crearNodo(int n,nodo *padre)
{
    nodo *nuevo= new nodo();
    nuevo->dato=n;
    nuevo->der=nullptr;
    nuevo->izq=nullptr;
    nuevo->padre=padre;
    return nuevo;
}

void abb::insertar(nodo *&arbol,int n,nodo *padre)
{
    if(arbol==nullptr)
    {
        nodo *nuevo=crearNodo(n,padre);
        arbol=nuevo;
    }
    else
    {
        int raiz=arbol->dato;
        if(n<raiz)
        {
            insertar(arbol->izq,n,arbol);
        }
        else
        {
            insertar(arbol->der,n,arbol);
        }
    }
}
```

- Capturas



```
<<MENU>>
1. Insertar Dato
2. Mostrar Arbol
3. Mostrar InOrden
4. Eliminar dato
5. Guardar
6. Cargar
0. Salir
R= 1
Ingreso dato: 15
El dato 15 fue ingresado
Presione una tecla para continuar . . .
```

Insertamos el nodo raíz.

```
<<MENU>>
1. Insertar Dato
2. Mostrar Arbol
3. Mostrar InOrden
4. Eliminar dato
5. Guardar
6. Cargar
0. Salir
    R= 1

Ingrese dato: 24
El dato 24 fue ingresado

Presione una tecla para continuar . . .
```

Insertamos el resto del árbol.

## 2. Mostrar arbol.

Este método nos muestra el arbol insertado de forma ordenada.

- Código

```
void abb::mostrar(nodo *arbol,int cont)
{
    if(arbol==nullptr)
    {
        return;
    }
    else
    {
        mostrar(arbol->der,cont+1);
        for(int i=0; i<cont; i++)
        {
            cout<<" ";
        }
        cout<<arbol->dato<<endl;
        mostrar(arbol->izq,cont+1);
    }
}
```

- Capturas

```
<<MENU>>
1. Insertar Dato
2. Mostrar Arbol
3. Mostrar InOrden
4. Eliminar dato
5. Guardar
6. Cargar
0. Salir
    R= 2

Arbol
    24
  20
    18
  15
    17
    12
  9
    7
  6
    4
  3
    1

Presione una tecla para continuar . . .
```

ABB Creado.

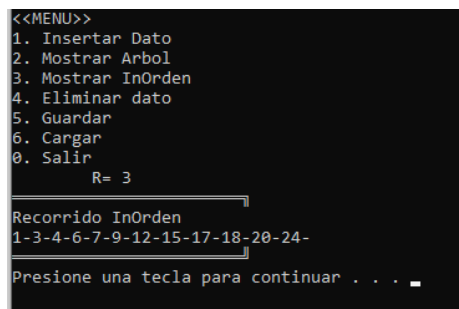
### 3. Mostrar InOrden.

Este método nos imprime el arbol de menor a mayor los datos.

- Código

```
void abb::inOrden(nodo *arbol)
{
    if(arbol==nullptr)
    {
        return;
    }
    else
    {
        inOrden(arbol->izq);
        cout<<arbol->dato<<"-";
        inOrden(arbol->der);
    }
}
```

- Capturas



```
<<MENU>>
1. Insertar Dato
2. Mostrar Arbol
3. Mostrar InOrden
4. Eliminar dato
5. Guardar
6. Cargar
0. Salir

R= 3

Recorrido InOrden
1-3-4-6-7-9-12-15-17-18-20-24-

Presione una tecla para continuar . . .
```

Impresión InOrden.

### 4. Eliminar dato.

Este método se encarga de eliminar un dato dentro del arbol, sin embargo, para realizar este método es necesario saber si tiene dos hijos, un hijo o es una raíz, para ello se realizan muchas validaciones y para cada una se usa una manera distinta de eliminar ya sea remplazando los datos por su siguiente menor o si es una hoja solamente eliminada el nodo hoja sin problema.

- Código

```
void abb::eliminar(nodo *arbol,int n)
{
    if(arbol==nullptr)
```

```

    {
        return;
    }
    else if(n<arbol->dato)
    {
        eliminar(arbol->izq,n);
    }
    else if(n>arbol->dato)
    {
        eliminar(arbol->der,n);
    }
    else
    {
        eliminarNodo(arbol);
    }
}

```

```

void abb::destruir(nodo *nodo)

```

```

{
    nodo->izq = nullptr;
    nodo->der = nullptr;

    delete nodo;
}

```

```

void abb::eliminarNodo(nodo *nodoEliminar)

```

```

{
    if(nodoEliminar->izq && nodoEliminar->der)
    {
        nodo *menor=minimo(nodoEliminar->der);
        nodoEliminar->dato=menor->dato;
        eliminarNodo(menor);
    }
    else if(nodoEliminar->izq)
    {
        remplazar(nodoEliminar,nodoEliminar->izq);
        destruir(nodoEliminar);
    }
    else if(nodoEliminar->der)
    {
        remplazar(nodoEliminar,nodoEliminar->der);
        destruir(nodoEliminar);
    }
    else
    {
        remplazar(nodoEliminar,nullptr);
        destruir(nodoEliminar);
    }
}

```

```

nodo *abb::minimo(nodo *arbol)
{

```

```

        if(arbol==nullptr)
        {
            return nullptr;
        }
        if(arbol->izq)
        {
            return minimo(arbol->izq);
        }
        else
        {
            return arbol;
        }
    }

void abb::reemplazar(nodo *arbol, nodo *nuevoNodo)
{
    if(arbol->padre)
    {
        if(arbol->dato == arbol->padre->izq->dato)
        {
            arbol->padre->izq = nuevoNodo;
        }
        else if(arbol->dato == arbol->padre->der->dato)
        {
            arbol->padre->der = nuevoNodo;
        }
    }
    if(nuevoNodo)
    {
        nuevoNodo->padre = arbol->padre;
    }
}

```

- Capturas

```

<<MENU>>
1. Insertar Dato
2. Mostrar Arbol
3. Mostrar InOrden
4. Eliminar dato
5. Guardar
6. Cargar
0. Salir

R= 4

Dato a eliminar: 6

Presione una tecla para continuar . . . _

```

Seleccionamos dato a eliminar, en este caso es un nodo con dos hijos.

```

Arbol
      24
     20
    18
   17
  15
  12
  9
 7
 4
3
1
Presione una tecla para continuar . . .

```

Comprobamos que fue eliminado.

```

<<MENU>>
1. Insertar Dato
2. Mostrar Arbol
3. Mostrar InOrden
4. Eliminar dato
5. Guardar
6. Cargar
0. Salir
    R= 4
Dato a eliminar: 18
Presione una tecla para continuar . . .

```

Seleccionamos dato a eliminar, en este caso es un nodo con un hijo.

```

      R= 2
Arbol
      24
     20
    17
  15
  12
  9
 7
 4
3
1
Presione una tecla para continuar . . .

```

Comprobamos que fue eliminado.

```

<<MENU>>
1. Insertar Dato
2. Mostrar Arbol
3. Mostrar InOrden
4. Eliminar dato
5. Guardar
6. Cargar
0. Salir
    R= 4
Dato a eliminar: 1
Presione una tecla para continuar . . .

```

Seleccionamos dato a eliminar, en este caso es un nodo hoja.



```
Arbol
  24
 20
 17
15  12
   9
   7  4
    3
Presione una tecla para continuar . . .
```

Comprobamos que fue  
eliminado.

## 5. Guardar.

Este método es el encargado de realizar un respaldo de datos en un archivo .txt el cual se guarda en nuestros archivos.

Para que este funcionara de manera correcta utilice la función ofstream y seleccione el nombre del archivo, después comprobamos que no genere error crear el archivo, enseguida pasamos al guardados de datos de manera ordenada para no perder el acomodo de nuestro arbol, esto se ejecuta hasta que nuestros apuntadores sean a nulo.

- Código

```
void abb::GuardarInformacion(nodo *actual)
{
    ofstream archivo( "file01.txt", ios::app );
    if(!archivo)
    {
        cerr << "No se pudo abrir el archivo" << endl;
        exit( EXIT_FAILURE );
    }
    if(actual)
    {
        while (actual)
        {
            GuardarInformacion(actual->izq);
            archivo << actual->num.getNum() << '|' << '*';
            GuardarInformacion(actual->der);
        }

        cout<<"Informacion guardada"<<endl;
    }
    else
    {
        cout<<"\nEl arbol esta vacia"<<endl;
    }
    archivo.close();
}
```

- Capturas

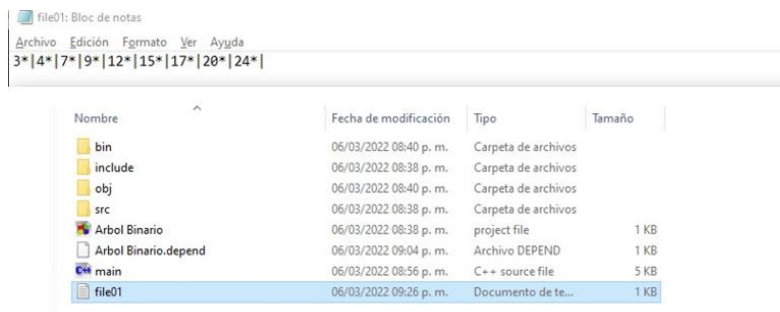
```
<<MENU>>
1. Insertar Dato
2. Mostrar Arbol
3. Mostrar InOrden
4. Eliminar dato
5. Guardar
6. Cargar
0. Salir

R= 5

Guardando datos...

Presione una tecla para continuar . . .
```

Seleccionamos la opción guardar.



Comprobamos que el archivo se creo y guardo correctamente.

## 6. Cargar.

Este método se apoya del método guardar, para cargar los datos respaldados para su uso.

Para que este funcionara se utilizó la función ifstream que a diferencia del ofstream este busca el archivo en nuestros documentos y lo lee cargando su información en nuestro programa.

Para esto declaramos variables que se utilizaran, después con ifstream nombramos el archivo a buscar enseguida se comprueba que no genere error abrir el archivo y después se empieza con la lectura dando ciertas declaraciones y comprobaciones y con archivo.read lee el documento por el método while recorre todo el documento convirtiendo los datos string en tipo entero por ultimo los guarda en nuestros datos los cuales son enlazados con el arbol.

- Código

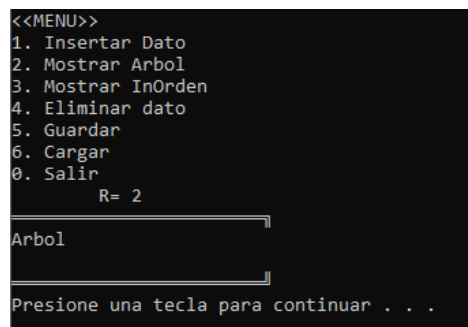
```
void abb::CargarInformacion()
{
    Data datoabb;
    int largo;
    string registro,num,buffer;
    ifstream archivo( "file01.txt", ios::in );
    if(!archivo)
    {
        cerr<<"No se pudo abrir el archivo"<<endl;
        exit( EXIT_FAILURE );
    }
    else
    {
        archivo.seekg(0,archivo.end);
        largo=archivo.tellg();
        archivo.seekg(0,archivo.beg);

        char *buffer= new char [largo];
        archivo.read(buffer,largo);
        stringstream ss(buffer);

        while(getline(ss,registro,'') and ss.eofbit)
        {
            stringstream ss_reg(registro);
            try
            {
                getline(ss_reg, num, '|');

                datoabb.setNum(stoi(num));
            }
            catch(invalid_argument const &e){}
            catch(out_of_range const &e){}
        }
        cout<<"Informacion cargada"<<endl;
    }
    archivo.close();
}
```

- Capturas



```
<<MENU>>
1. Insertar Dato
2. Mostrar Arbol
3. Mostrar InOrden
4. Eliminar dato
5. Guardar
6. Cargar
0. Salir
R= 2
Arbol
Presione una tecla para continuar . . .
```

Verificamos que el  
árbol esta vacio.

```

<<MENU>>
1. Insertar Dato
2. Mostrar Arbol
3. Mostrar InOrden
4. Eliminar dato
5. Guardar
6. Cargar
0. Salir
    R= 6

Cargando datos...

Presione una tecla para continuar . . .

```

Llamamos el método cargar.

```

<<MENU>>
1. Insertar Dato
2. Mostrar Arbol
3. Mostrar InOrden
4. Eliminar dato
5. Guardar
6. Cargar
0. Salir
    R= 2

Arbol
    24
   20
  17
15
  12
   9
   7
   4
   3

Presione una tecla para continuar . . .

```

Comprobamos que el árbol se cargara correctamente.

## CONCLUSIÓN

Los ABB son muy efectivos para organizar datos y sus métodos principales son sencillos de programar, sin embargo, el método eliminar es uno de los mas complejos ya que sus validaciones deben comprobar todo el árbol para reemplazar o simplemente eliminar, fue difícil de programar, pero con la investigación adecuada pude lograr dichos métodos.

Los métodos guardar y cargar se generan de forma distinta a otras TDA ya que al guardarlos deben ser organizados para el momento de cargar el árbol siga cumpliendo su función, aun así, con una investigación extensa pude programar y comprobar los métodos.