

Contenido

| | |
|---|-----------|
| 1 Explicación del programa | 2 |
| 2 Cómo se organizo el TDA | 2 |
| 3 Estrategia de almacenamiendo y recuperación de datos | 3 |
| 4 Explicación de las funciones | 3 |
| 4.1 Insertar | 3 |
| 4.2 Imprimir inorden | 4 |
| 4.3 Imprimir posorden | 4 |
| 4.4 Imprimir preorden | 4 |
| 4.5 Eliminar nodo | 5 |
| 4.6 Eliminar árbol | 6 |
| 4.7 Buscar | 7 |
| 4.8 Leer archivo | 7 |
| 4.9 Guardar archivo | 9 |
| 4.10Salir | 10 |
| 5 Capturas de pantalla | 13 |
| 6 Conclusiones | 19 |

1 Explicación del programa

El programa incorpora un árbol de búsqueda binaria (ABB) en C++, haciendo uso de memoria dinámica (apuntadores) y de manejo de archivos utilizando delimitadores.

Las operaciones codificadas para el árbol fueron:

- Insertar
- Imprimir inorden
- Imprimir posorden
- Imprimir preorden
- Eliminar nodo
- Eliminar árbol
- Buscar
- Leer archivo
- Guardar archivo
- Salir

2 Cómo se organizo el TDA

El nodo que contiene el tipo de dato *Student* contiene tres atributos:

- name
- age
- major

Al insertar un nuevo nodo al árbol el programa pedirá estos tres datos, añadiéndolos al final como un solo tipo de dato (recuerdos de Datos 1).

3 Estrategia de almacenamiento y recuperación de datos

4 Explicación de las funciones

4.1 Insertar

```
void Tree::insert(Student data){
    if(root == nullptr){
        root = new TreeNode(data);
    }else{
        TreeNode* current = root;
        while(true){
            if(data.name < current->data.name){
                if(current->left == nullptr){
                    current->left = new TreeNode(data);
                    break;
                }else{
                    current = current->left;
                }
            }else{
                if(current->right == nullptr){
                    current->right = new TreeNode(data);
                    break;
                }else{
                    current = current->right;
                }
            }
        }
        std::cout << "\nAlumno agregado" << std::endl;
    }
}
```

- Recibe el tipo de dato *Student*
- Devuelve el nodo *root* si es el primer nodo insertado en el árbol
- Si no, recorre el árbol por la izquierda comparando el dato *name* con el nodo *current* hasta asignarlo
- Si el nombre es “mayor”, recorre por la derecha hasta encontrar su lugar

4.2 Imprimir inorden

```
void Tree::traverseInOrder(TreeNode* node){
    if(node != nullptr){
        traverseInOrder(node->left);
        std::cout << node->data.name
                    << " ("
                    << node->data.age
                    << ", "
                    << node->data.major
                    << ")" << std::endl;
        traverseInOrder(node->right);
    }
}
```

- Mientras el nodo sea distinto de nulo, recorre la parte izquierda
- Llegando al tope, imprime los datos del nodo visitado
- Recorre la parte izquierda

4.3 Imprimir posorden

```
void Tree::traversePostOrder(TreeNode* node){
    if(node != nullptr){
        traversePostOrder(node->left);
        traversePostOrder(node->right);
        std::cout << node->data.name
                    << " ("
                    << node->data.age
                    << ", "
                    << node->data.major
                    << ")" << std::endl;
    }
}
```

- Se mueve a la izquierda
- Se mueve a la derecha
- Llegando al tope, imprime el nodo visitado

4.4 Imprimir preorden

```
void Tree::traversePreOrder(TreeNode* node){
    if(node != nullptr){
```

```

        std::cout << node->data.name
                    << " ("
                    << node->data.age
                    << ", "
                    << node->data.major
                    << ")" << std::endl;
        traversePreOrder(node->left);
        traversePreOrder(node->right);
    }
}

```

- Imprime el nodo visitado
- Se mueve a la izquierda
- Se mueve a la derecha

4.5 Eliminar nodo

```

void Tree::deleteNode(TreeNode* root, std::string& name){
    if (root == nullptr){
        return;
    }

    if(root->data.name == name){
        if(root->left == nullptr && root->right == nullptr){
            delete root;
            root = nullptr;
        }else if(root->left == nullptr){
            TreeNode* temp = root;
            root = root->right;
            temp->right = nullptr;
            delete temp;
        }else if(root->right == nullptr){
            TreeNode* temp = root;
            root = root->left;
            temp->left = nullptr;
            delete temp;
        }else{
            TreeNode* temp = root->right;
            while(temp->left != nullptr){
                temp = temp->left;
            }
        }
    }
}

```

```

        root->data.name = temp->data.name;
        deleteNode(root->right, temp->data.name);
    }
} else if(name < root->data.name){
    deleteNode(root->left, name);
} else{
    deleteNode(root->right, name);
}
}
}

```

- Busca el nodo a eliminar en el lado izquierdo y derecho del árbol
- Si el nodo es encontrado con el nombre específico
 - Verifica si tiene un hijo, dos o ninguno
 - * Si tiene ninguno elimina el nodo
 - * Si tiene uno, el padre del nodo visitado apuntará al hijo del nodo, esté en la izquierda o en la derecha
 - * Si tiene dos, mientras el nodo sea distinto de nulo, se recorre a la izquierda, moviendo los datos a su consiguiente, y eliminando el más a la derecha

4.6 Eliminar árbol

```

void Tree::deleteAll(TreeNode* root){
    if(root == nullptr){
        return;
    }

    deleteAll(root->left);
    deleteAll(root->right);

    delete root;
    root = nullptr;
}

```

- Si el nodo es nulo, regresa
- Si no, llama recursivamente al lado izquierdo del nodo
- Al igual que el derecho
- Eliminando el nodo visitado
- Para terminar aterrizándolo

4.7 Buscar

```
void Tree::search(TreeNode* root, std::string name){
    if(root == nullptr){
        return;
    }
    search(root->left, name);
    if(root->data.name == name){
        std::cout << "Alumno encontrado:" << std::endl;
        std::cout << root->data.name
                    << ", " << root->data.age
                    << ", " << root->data.major << std::endl;
    }
    search(root->right, name);
}
```

- Si el nodo es nulo, regresa, indicando que no se encontró
- Si no, busca por la izquierda
 - Si el nodo actual corresponde al nombre a buscar, lo imprime
- Si no, busca por la izquierda

4.8 Leer archivo

Utiliza dos funciones

```
TreeNode* Tree::readFile(const std::string filename,
                        const char fDel, const char rDel){

    std::ifstream file;
    file.open(filename);

    TreeNode* root = nullptr;

    readFileHelper(root, file, fDel, rDel);

    std::cout << "Archivo leído" << std::endl;

    return root;
}
```

Que simplemente abre el archivo con los parámetros *filename*, crea un nodo raíz y llama a *readFileHelper* con los delimitadores de campo y registro

Cabe aclarar que los delimitadores utilizados son:

- , para campos
- **Carácter de salto de línea** para registros

Haciendo que el archivo *file01.txt* termine de la siguiente manera:

- Diego,21,Ingeniería
- María,22,Contaduría
- Naomi,21, Veterinaria

Continuando con la función *readFileHelper*

```
void Tree::readFileHelper(TreeNode*& node,
                           std::ifstream& file,
                           const char fDel,
                           const char rDel){

    std::string name;
    // Necesario para leerlo del archivo,
    // de ahí lo convertimos a int
    std::string ageStr;
    std::string major;

    getline(file, name, fDel);
    getline(file, ageStr, fDel);
    getline(file, major, rDel);

    if(name.empty() || ageStr.empty() || major.empty())
        node = nullptr;
    else{
        int age = stoi(ageStr);

        // Leemos de la misma manera en que leemos
        // el archivo
        node = new TreeNode(Student(name, age, major));
        readFileHelper(node->left, file, fDel, rDel);
        readFileHelper(node->right, file, fDel, rDel);
    }
}
```

Se crean 3 variables para retomar los datos del archivo

- name
- ageStr
- major

Siendo todos *string*, a pesar que *age* debería ser *int*, el archivo de texto no lo reconoce como entero, por lo tanto, es necesario cambiarlo a su dato correspondiente en memoria.

- Verifica si uno de los campos está vacío (por si pasa lo peor), y crea el nodo en vacío.
- Si no, hace el cambio a *int* para la variable *age*
- Crea un nuevo nodo con todos los datos obtenidos
- Llama a la función del lado izquierdo y derecho

La forma en la que lee el archivo es de la misma manera en la que la impresión preorden, ¡importante para poder tener los datos en orden!

4.9 Guardar archivo

De la misma manera para leer el archivo, la forma que guarda es en dos funciones

```
bool Tree::saveFile(TreeNode* root, const std::string& filename,
                        const char fDel, const char rDel){

    std::ofstream file;
    file.open(filename);

    saveFileHelper(root, file, fDel, rDel);

    file.close();
    // Para changes, false para saber que no hay
    // changes, jeje, ¿entiendes?
    return false;
}
```

- Crea un archivo con el nombre especificado
- Llama a

```

void Tree::saveFileHelper(TreeNode* node, std::ofstream& file,
                        const char fDel, const char rDel){

    if(node == nullptr){
        return;
    }

    // Recorrido preorden PREORDEN
    // Guarda el nodo
    // Se va al izquierdo y derecho
    file << node->data.name << fDel
        << node->data.age << fDel
        << node->data.major << rDel;

    saveFileHelper(node->left, file, fDel, rDel);
    saveFileHelper(node->right, file, fDel, rDel);
}

```

- Añade cada uno de los datos conforme a los delimitadores de campo y registro mencionados
- Llama a la función por la derecha y por la izquierda

4.10 Salir

Digo, es...intuitivo, ¿no?

```

do{
    option = choose();
    switch(option){
        // Insertar
        case '1':{
            readInput();
            oak.insert(Student(student.name,
                               student.age,
                               student.major));

            break;
        }
        // Inorden
        case '2':{
            if(oak.isTreeEmpty(oak.root))
                std::cout << "No existe" << std::endl;
            else{

```

```

        std::cout << &oak.root << std::endl;
        shoutItOut();
        oak.traverseInOrder();
    }
    break;
}
// Posorden
case '3':{
    if(oak.isTreeEmpty(oak.root))
        std::cout << "No existe" << std::endl;
    else{
        shoutItOut();
        oak.traversePostOrder();
    }
    break;
}
// Preorden
case '4':{
    if(oak.isTreeEmpty(oak.root))
        std::cout << "No existe" << std::endl;
    else{
        shoutItOut();
        oak.traversePreOrder();
    }
    break;
}
// Eliminar nodo
case '5':{
    std::string lookfor;
    std::cin.ignore();
    std::cout << "Nombre a eliminar: ";
    getline(std::cin, lookfor);
    oak.deleteNode(oak.root, lookfor);
    break;
}
// Eliminar árbol
case '6':{
    oak.deleteAll(oak.root);
    oak.root = nullptr;
    break;
}
// Buscar

```

```

case '7':{
    std::cin.ignore();
    std::string lookfor;
    std::cout << "Nombre a buscar: ";
    getline(std::cin, lookfor);
    oak.search(oak.root, lookfor);
    break;
}
// Leer archivo
case '8':{
    // Haz que elimine lo que está
    // anteriormente para así no gastar
    // memoria
    oak.root = oak.readFile(filename, fDel, rDel);
    break;
}
// Guardar archivo
case '9':{
    char subOption;
    if(!oak.isFileEmpty(filename)){
        std::cout << "El archivo tiene "
                    << "contenido, "
                    << "¿sobreescribir?\n"
                    << "1. Sí 2. No: ";
        std::cin >> subOption;
        switch(subOption){
            case '1':{
                changes = oak.saveFile(oak.
                                     file
                                     fDeT
                                     rDeT

                break;
            }
            case '2':{
                std::cout << "Saliendo"
                          << std::endl;

                break;
            }
            default:
                std::cout << "Inválido,"
                          << "saliendo"
                          << std::endl;
        }
    }
}

```

```

    }

    }else{
        changes = oak.saveFile(oak.root,
                                filename,
                                fDel,
                                rDel);
    }
    break;
}
case 'S':{
    std::cout << "Adiós" << std::endl;
    break;
}
case 's':{
    std::cout << "Adiós" << std::endl;
    break;
}
default:
    std::cout << "Opción incorrecta" << std::endl;
}

}while(option != 's' && option != 'S');

```

Tanto código para una función sencilla que no cabe en el documento. Es el menú en el main, la opción de salida es s.


5 Capturas de pantalla

Programa recién abierto

Árbol ABB

1. Insertar
 2. Imprimir inorden
 3. Imprimir posorden
 4. Imprimir preorden (recomendado)
 5. Eliminar nodo
 6. Eliminar árbol
 7. Buscar
 8. Leer archivo
 9. Guardar archivo
 - S. Salir
- :

Insertando un dato



```
: 1

Nombre: Diego
Edad: 21
Carrera: Ingeniería

Alumno agregado

(El árbol ha sido modificado, recomendable guardar)
1. Insertar
2. Imprimir inorden
3. Imprimir posorden
4. Imprimir preorden (recomendado)
5. Eliminar nodo
6. Eliminar árbol
7. Buscar
8. Leer archivo
9. Guardar archivo
S. Salir
:
```

El programa muestra si se modificó el árbol
Imprimiendo los datos



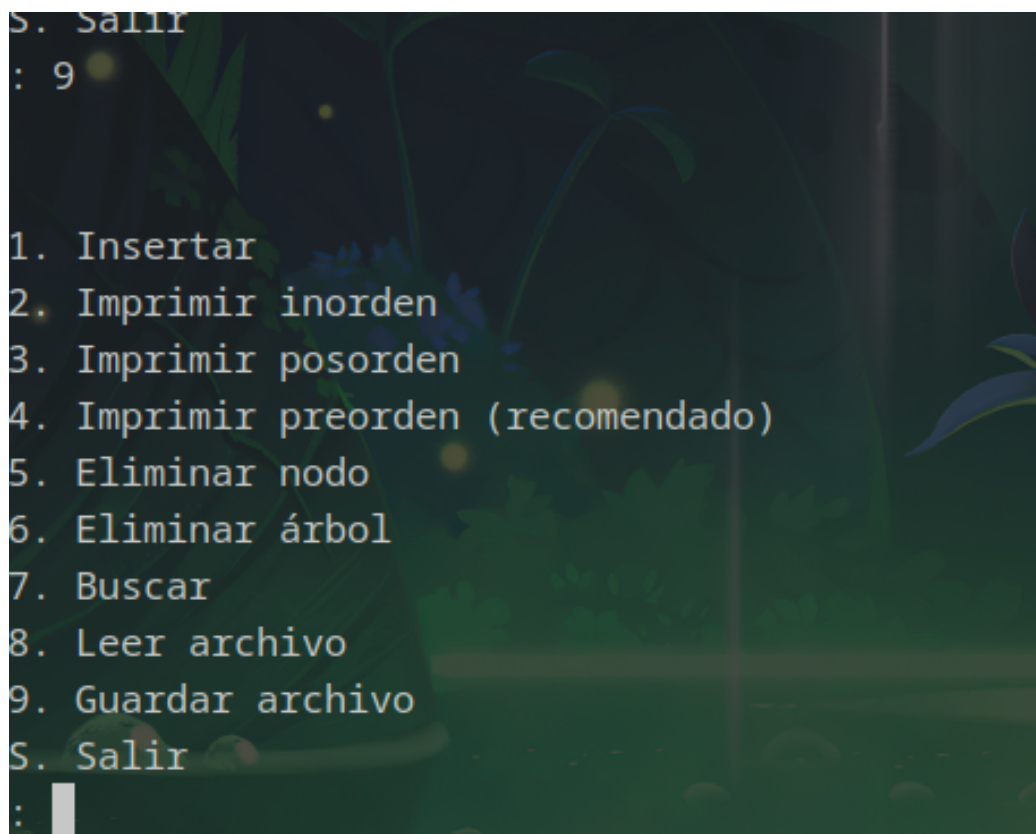
```
: 3

Nombre (edad, carrera)

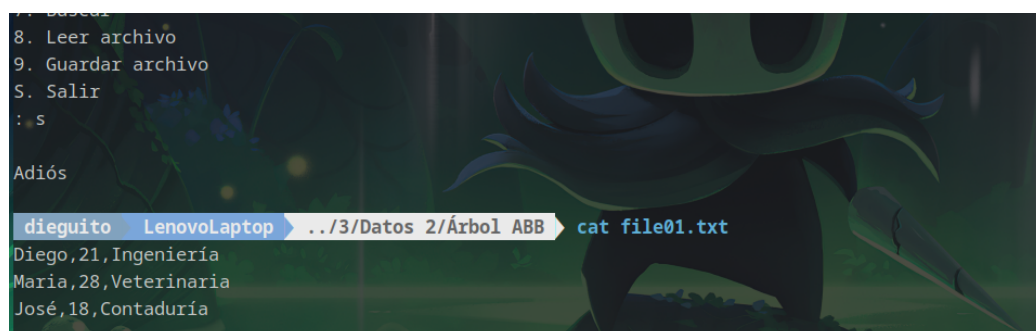
José (18, Contaduría)
Maria (28, Veterinaria)
Diego (21, Ingeniería)

(El árbol ha sido modificado, recomendable guardar)
1. Insertar
2. Imprimir inorden
3. Imprimir posorden
4. Imprimir preorden (recomendado)
5. Eliminar nodo
6. Eliminar árbol
7. Buscar
8. Leer archivo
9. Guardar archivo
S. Salir
:
```

Guardando a archivo



Mostrando el archivo guardado, mostrando los campos separados por los delimitadores previamente mencionados



Reabriendo el archivo y cargándolo a memoria

```
dieguito  LenovoLaptop  ../3/Datos 2/Árbol ABB  ./tree
Árbol ABB

1. Insertar
2. Imprimir inorden
3. Imprimir posorden
4. Imprimir preorden (recomendado)
5. Eliminar nodo
6. Eliminar árbol
7. Buscar
8. Leer archivo
9. Guardar archivo
5. Salir
: 8

Archivo leído

1. Insertar
2. Imprimir inorden
3. Imprimir posorden
4. Imprimir preorden (recomendado)
5. Eliminar nodo
6. Eliminar árbol
7. Buscar
8. Leer archivo
9. Guardar archivo
5. Salir
: 
```

Mostrando que sí se guardó

```
0x555c0b8c03c0
Nombre (edad, carrera)

José (18, Contaduría)
Maria (28, Veterinaria)
Diego (21, Ingeniería)

1. Insertar
2. Imprimir inorden
3. Imprimir posorden
4. Imprimir preorden (recomendado)
5. Eliminar nodo
6. Eliminar árbol
7. Buscar
8. Leer archivo
9. Guardar archivo
5. Salir
:
```

6 Conclusiones

Me partí el coco intentando entender los árboles, no tengo más opinión sobre esto, esperando a los grafos con miedo.