

13-4-2019

# EL ÁRBOL BINARIO DE BÚSQUEDA, IMPLEMENTACIÓN DINÁMICA



# CUCEI

david gutierrez alvarez

## RESUMEN PERSONAL Y FORMA DE ABORDAR EL PROBLEMA

A pesar de que este trabajo no fue algo muy fácil el echo de ya tener conocimientos y practica no se me dificulto a la hora de implementarlo, aun me falta mejorar en mi forma de codificar pero con todas las practicas que hemos tenido ya puedo hacer algo decente

## Main.cpp

```
#include <iostream>
#include "menu.h"

using namespace std;

int main() {
    Menu menu;

    menu.selectOption();

    return 0;
}
```

## Menú.h

```
#ifndef MENU_H
#define MENU_H

#include <iostream>
#include "BTree.h"

class Menu {
private:
    BTree tree;

    enum option{
        generate = 1,
        preorder,
        inorder,
        postorder,
        close
    };

public:
    Menu();

    long random();

    void generated(size_t i4);

    void selectOption();

};

#endif // MENU_H
```

## Menú.cpp

```
#include "menu.h"
#include <random>
#include <chrono>

using std::cout;
using std::cin;
using std::endl;

Menu::Menu() { }
```



```

#include <iostream>
using namespace std;

class BTree {
private:
    struct Node {
        int i;
        Node *left;
        Node *right;
    };
    Node *root;

    void clear (Node *&n) {
        if(n != nullptr) {
            clear(n->left);
            clear(n->right);
            delete n;
        }
    }

    int insert(Node *&r, Node *&n) {
        if(r == nullptr) {
            r = n;
            return 0;
        }
        if(r->i > n->i) {
            if(!r->left) {
                r->left = n;
                return 0;
            }
            if(r->left) {
                return insert(r->left, n);
            }
        }

        if(r->i <= n->i) {
            if(!r->right) {
                r->right = n;
                return 0;
            }
            if(r->right) {
                return insert(r->right, n);
            }
        }
        return -1;
    }

    void inPrint(Node *n) const {
        if(n != nullptr) {
            inPrint(n->left);
            cout << n->i << endl;
            inPrint(n->right);
        }
    }

    void prePrint(Node *n) const {
        if(n != nullptr) {
            cout << n->i << endl;
            prePrint(n->left);
            prePrint(n->right);
        }
    }

    void postPrint(Node *n) const {
        if(n != nullptr) {

```

```

        postPrint(n->right);
        cout << n->i << endl;
        postPrint(n->left);
    }
}

bool find (Node *n, int i) const {
    if(n->i == i) {
        return 1;
    }

    if(n->i <= i) {
        if(n->right) {
            return find(n->right,i);
        }
        if(!n->right) {
            return 0;
        }
    }

    if(n->i > i) {
        if(n->left) {
            return find(n->left,i);
        }
        if(!n->right) {
            return 0;
        }
    }
}

int count(Node *n) const {
    if(n) {
        return 1+ (count(n->left))+(count(n->right));
    }
    if(!n) {
        return 0;
    }
}

int seek(Node *&n , int i) {
    if(n->i == i) {
        destroy(n); // call destroy
        return 0;
    }
    if(n->i <= i) {
        if(n->right) {
            return seek(n->right, i);
        }
        if(!n->left) {
            return -1;
        }
    }
    if(n->i > i) {
        if(n->left) {
            return seek(n->left, i);
        }
        if(!n->left) {
            return -1;
        }
    }
    return -1;
}

void destroy(Node *&n) {
    if(!n->left && !n->right) {

```

```

        delete n;
        n = NULL;
    } else {
        Node *l;
        Node *r;
        l = n->left;
        r = n->right;
        delete n;
        n = NULL;
        copy(l);
        copy(r);
    }
}

void copy(Node *n) {
    if(n != nullptr) {
        insert(root, n);
    }
}

public:
    BTree() : root(nullptr) {}

    BTree(const BTree &rhs) {
        root = nullptr;
        operator=(rhs);
    }

    ~BTree() {
        if(root) {
            clear();
            delete root;
        }
    }

    BTree& operator = (const BTree &rhs) {
        if(root) {
            clear(root);
            root = NULL;
        }

        Node *temp;
        Node *nerd;
        temp = rhs.root;
        if(!temp) {
            return *this;
        }
        while(temp) {
            if(!temp->left) {
                insert(temp->i);
                temp = temp->right;
            } else {
                nerd = temp->left;
                while(nerd->right && nerd->right != temp) {
                    nerd = nerd->right;
                }

                if(!nerd->right) {
                    nerd->right = temp;
                    temp = temp->left;
                } else {
                    nerd->right = NULL;
                    insert(temp->i);
                    temp = temp->right;
                }
            }
        }
    }

```

```

    }
}
return *this;
}

bool operator == (const BTree &rhs) {
    int red[2];
    red[0] = count(root);
    red[1] = count(rhs.root);
    if(red[0] == red[1]) {
        return 1;
    }
    return 0;
}

bool operator < (const BTree &rhs) {
    int red[2];

    red[0] = count(root);
    red[1] = count(rhs.root);

    if(red[0] < red[1]) {
        return 1;
    }
    return 0;
}

bool isFull() {
    if(root != nullptr) {
        return 1;
    }
    if(!root)
    {
        return 0;
    }
}

bool isEmpty() {
    if(root != nullptr) {
        return false;
    }
    return false;
}

void clear() {
    if(root != nullptr) {
        clear(root);
        root = NULL;
    }
}

int insert(int i) {
    Node *temp;
    temp = new Node;
    temp->i = i;
    temp->left = nullptr;
    temp->right = nullptr;
    return insert(root, temp);
}

bool find(int i) const {
    if(root) {
        return find(root, i);
    }
    if(!root) {
        return 0;
    }
}

```



```

    }
}

int remove (int i) {
    if(root != nullptr) {
        return seek(root,i);
    }
    return -1;
}

void inPrint() const {
    if(root != nullptr) {
        inPrint(root);
        cout << endl;
    }
}

void prePrint() const {
    if(root != nullptr) {
        prePrint(root);
        cout << endl;
    }
}

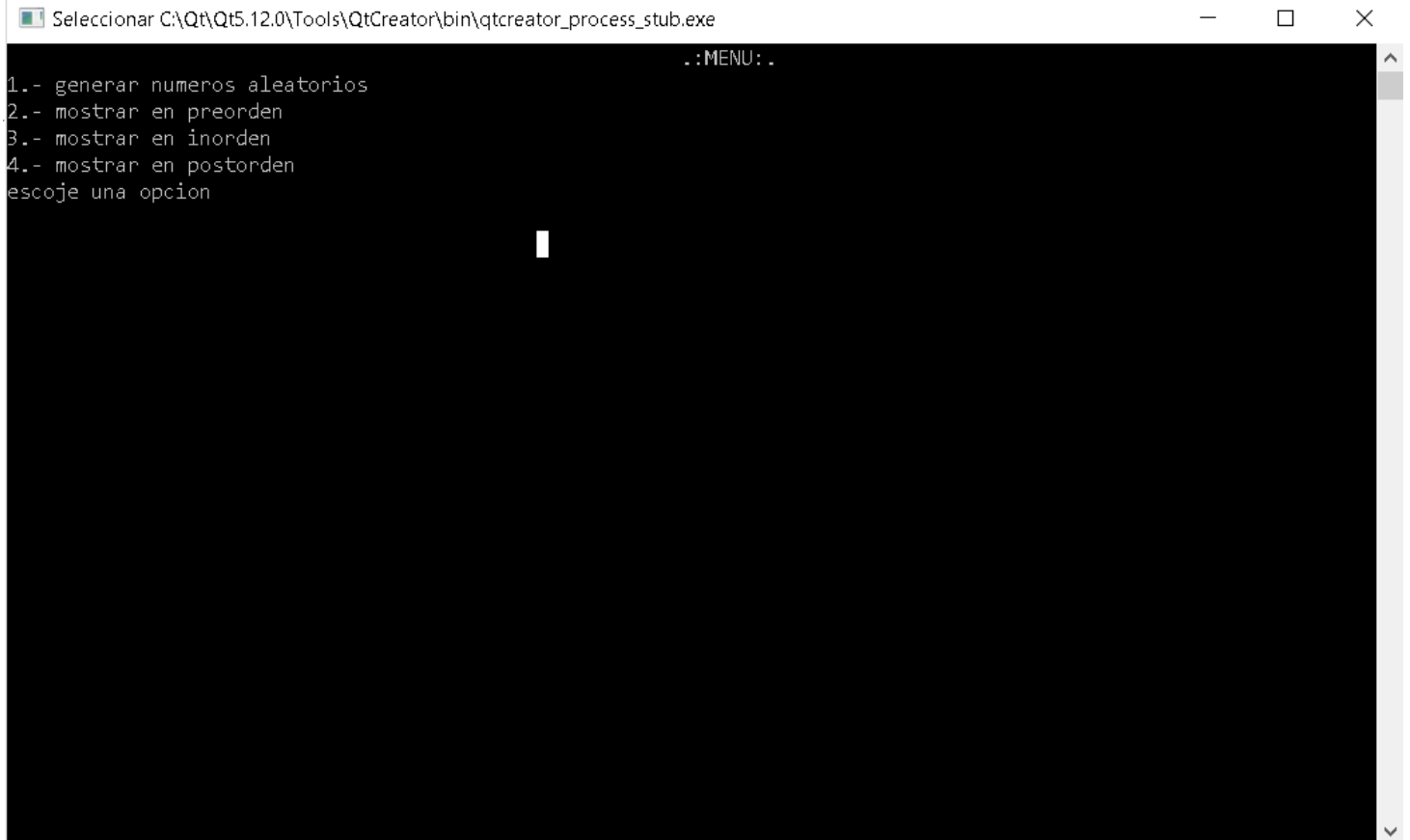
void postPrint() const {
    if(root != nullptr) {
        postPrint(root);
        cout << endl;
    }
}

};

#endif // BTREE_H

```

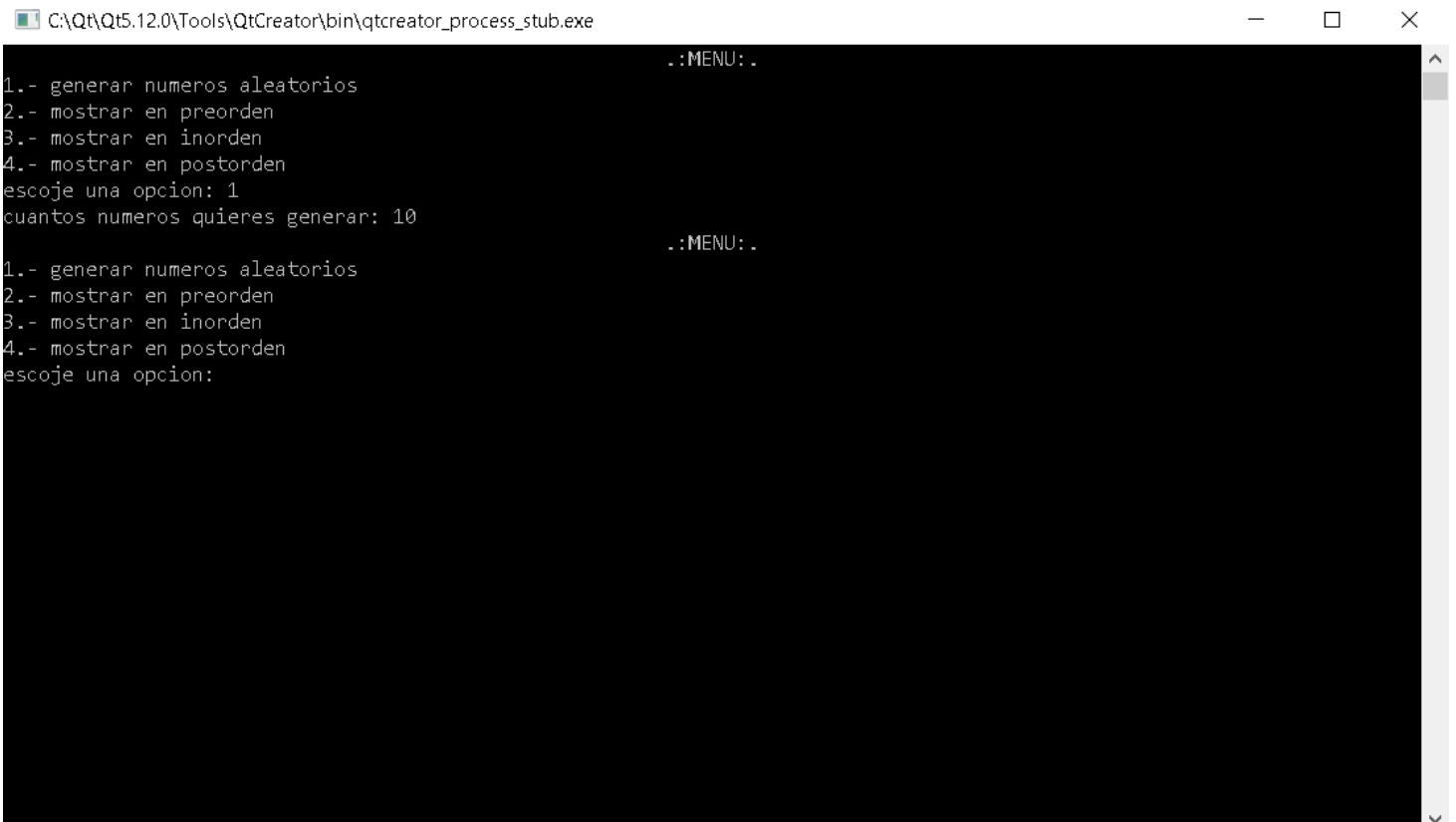
# CAPTURAS DE PANTALLA



Seleccionar C:\Qt\Qt5.12.0\Tools\QtCreator\bin\qtcreator\_process\_stub.exe

```
..:MENU:..  
1.- generar numeros aleatorios  
2.- mostrar en preorden  
3.- mostrar en inorden  
4.- mostrar en postorden  
escoje una opcion  
█
```

## Menú



C:\Qt\Qt5.12.0\Tools\QtCreator\bin\qtcreator\_process\_stub.exe

```
..:MENU:..  
1.- generar numeros aleatorios  
2.- mostrar en preorden  
3.- mostrar en inorden  
4.- mostrar en postorden  
escoje una opcion: 1  
cuantos numeros quieres generar: 10  
..:MENU:..  
1.- generar numeros aleatorios  
2.- mostrar en preorden  
3.- mostrar en inorden  
4.- mostrar en postorden  
escoje una opcion:
```

Aquí se generaron 10 numeros random

```
C:\Qt\Qt5.12.0\Tools\QtCreator\bin\qtcreator_process_stub.exe

.:MENU:.
1.- generar numeros aleatorios
2.- mostrar en preorden
3.- mostrar en inorden
4.- mostrar en postorden
escoje una opcion: 1
cuantos numeros quieres generar: 10

.:MENU:.
1.- generar numeros aleatorios
2.- mostrar en preorden
3.- mostrar en inorden
4.- mostrar en postorden
escoje una opcion: 2
770615
785222
788870
792526
796173
799829
803477
807133
810781
814437

.:MENU:.
1.- generar numeros aleatorios
2.- mostrar en preorden
3.- mostrar en inorden
4.- mostrar en postorden
escoje una opcion:
```

Vista en preorden

```
C:\Qt\Qt5.12.0\Tools\QtCreator\bin\qtcreator_process_stub.exe

796173
799829
803477
807133
810781
814437

.:MENU:.
1.- generar numeros aleatorios
2.- mostrar en preorden
3.- mostrar en inorden
4.- mostrar en postorden
escoje una opcion: 3
770615
785222
788870
792526
796173
799829
803477
807133
810781
814437

.:MENU:.
1.- generar numeros aleatorios
2.- mostrar en preorden
3.- mostrar en inorden
4.- mostrar en postorden
escoje una opcion:
```

## Vista en inorden

```
C:\Qt\Qt5.12.0\Tools\QtCreator\bin\qtcreeator_process_stub.exe

796173
799829
803477
807133
810781
814437

.:MENU:.

1.- generar numeros aleatorios
2.- mostrar en preorden
3.- mostrar en inorden
4.- mostrar en postorden
escoje una opcion: 4
814437
810781
807133
803477
799829
796173
792526
788870
785222
770615

.:MENU:.

1.- generar numeros aleatorios
2.- mostrar en preorden
3.- mostrar en inorden
4.- mostrar en postorden
escoje una opcion:
```

## Vista en postorden