

```
1 #include "bintree.h"
2 #include <iostream>
3
4 #include <queue>
5 #include <cmath>
6 #include <iomanip>
7
8 using namespace std;
9
10 template <class T>
11 void MostrarArbol(const bintree<T> &A, typename bintree<T>::node root)
12 {
13     queue<typename bintree<T>::node> colaNodos;
14     int totalNodos = A.size();
15     int techo = log2(totalNodos + 1);
16     colaNodos.push(root);
17     int pot = 0;
18     while (colaNodos.size() > 0)
19     {
20         int niveles = colaNodos.size();
21         while (niveles > 0)
22         {
23             typename bintree<T>::node nodoAux = colaNodos.front();
24             colaNodos.pop();
25             cout << setw((niveles == pow(2, pot)) ? pow(2, (techo - pot)) : pow(2,
26 (techo - pot + 1)));
27             cout << *nodoAux;
28             if (!nodoAux.left().null())
29                 colaNodos.push(nodoAux.left());
30             if (!nodoAux.right().null())
31                 colaNodos.push(nodoAux.right());
32             niveles--;
33         }
34         pot++;
35         cout << endl;
36     }
37 }
38 double valor(char a)
39 {
40     return (double)a;
41 }
42
43 double evaluacion(bintree<char> arb, bintree<char>::node v)
44 {
45     if (!v.null())
46     {
47         switch (*v)
48         {
49             case '+':
50                 return evaluacion(arb,v.left()) + evaluacion(arb,v.right());
51                 break;
52             case '*':
53                 return evaluacion(arb,v.left()) * evaluacion(arb,v.right());
54                 break;
55             case '-':
56                 return evaluacion(arb,v.left()) - evaluacion(arb,v.right());
57                 break;
58             case '/':
59                 return evaluacion(arb,v.left()) / evaluacion(arb,v.right());
```

```
60         break;
61     default:
62         return valor(*v);
63     break;
64 }
65 }
66 }
67
68 int main()
69 {
70     bintree<char> arb('*');
71
72     bintree<char>::node aux;
73
74     //RAMA IZQUIERDA
75
76     arb.insert_left(arb.root(), '-');
77
78     aux = arb.root().left();
79
80     arb.insert_left(aux, '/');
81
82     arb.insert_right(aux, '+');
83
84     aux = aux.left();
85
86     arb.insert_left(aux, 'A');
87
88     arb.insert_right(aux, 'C');
89
90     aux = aux.parent().right();
91
92     arb.insert_left(aux, 'D');
93
94     arb.insert_right(aux, 'Z');
95
96     //RAMA DERECHA
97
98     arb.insert_right(arb.root(), '*');
99
100     aux = arb.root().right();
101
102     arb.insert_left(aux, '/');
103
104     arb.insert_right(aux, '+');
105
106     aux = aux.left();
107
108     arb.insert_left(aux, 'Y');
109
110     arb.insert_right(aux, 'H');
111
112     aux = aux.parent().right();
113
114     arb.insert_left(aux, 'F');
115
116     arb.insert_right(aux, 'F');
117
118     cout << "-----ARBOL ORIGINAL-----" << endl;
119 }
```

```
120 |   MostrarArbol(arb, arb.root());  
121 |  
122 |   cout << "El valor del arbol es --> " << evaluacion(arb, arb.root()) << endl;  
123 | }
```