LAB - 11

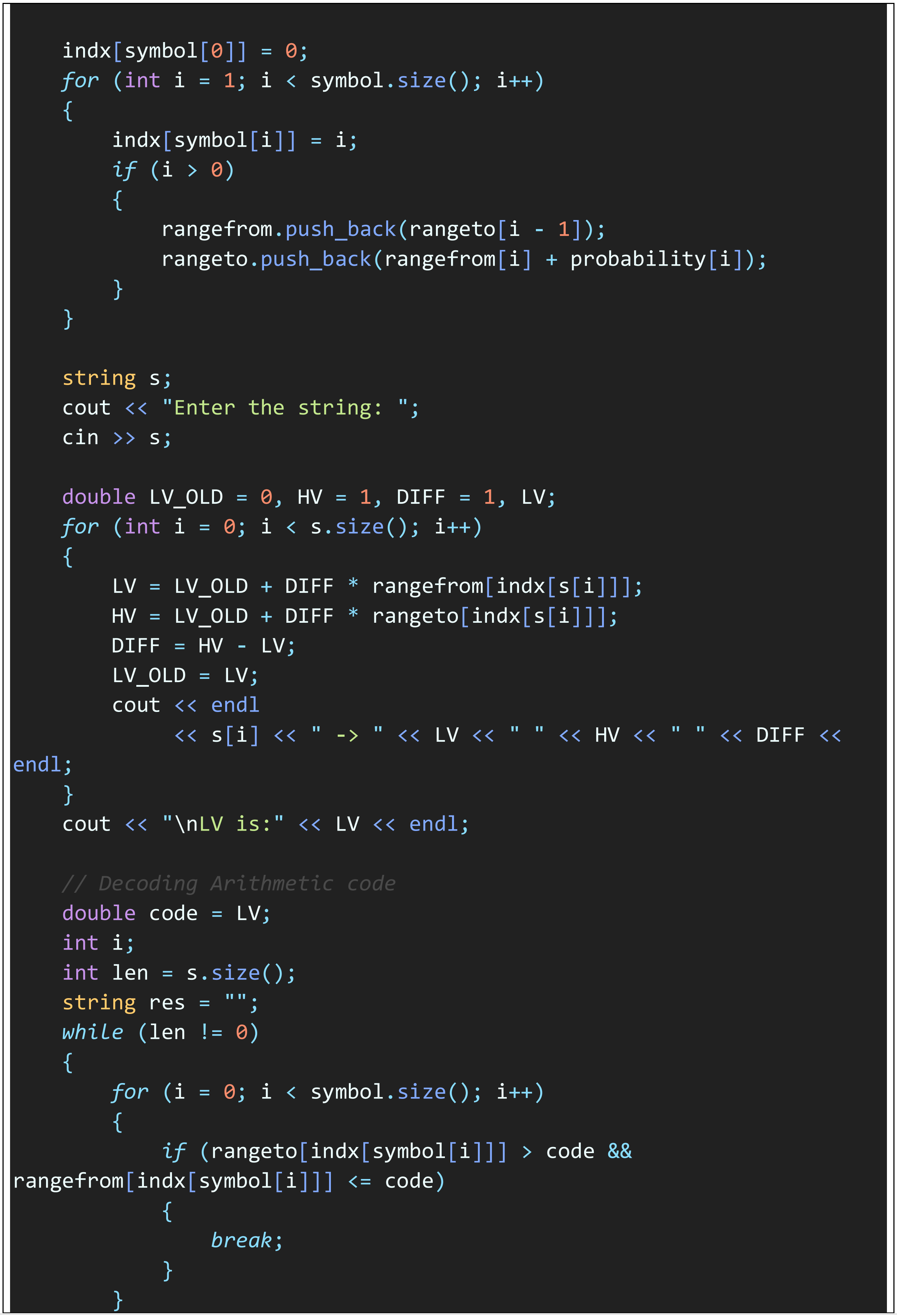
Name	Keval D Gandevia
Roll Number	CE046
ID	19CEUEG017
Subject	Image Processing

Aim: Implement basic compression techniques.

Q. 1: Implement Arithmetic Coding and Decoding. A). Take the data set given in the pdf and find the codewords for GERMAN and FRANCE. B). Decode the words from their respective codewords.

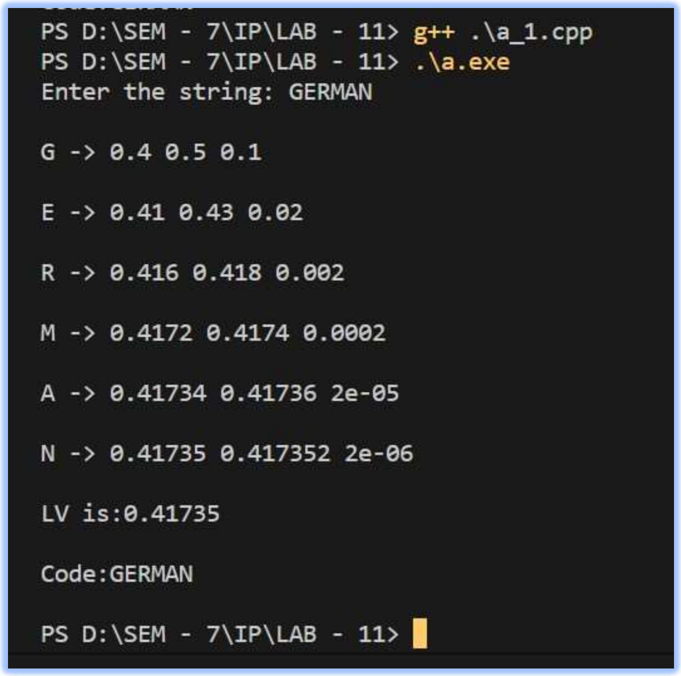
- **Code:**

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    vector<char> symbol = {'Y', 'E', 'R', 'G', 'N', 'M', 'A', 'F', 'C'};
    vector<double> probability = {0.1, 0.2, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1};
    unordered_map<char, int> indx;
    vector<double> rangefrom = {0.0};
    vector<double> rangeto = {probability[0]};
```

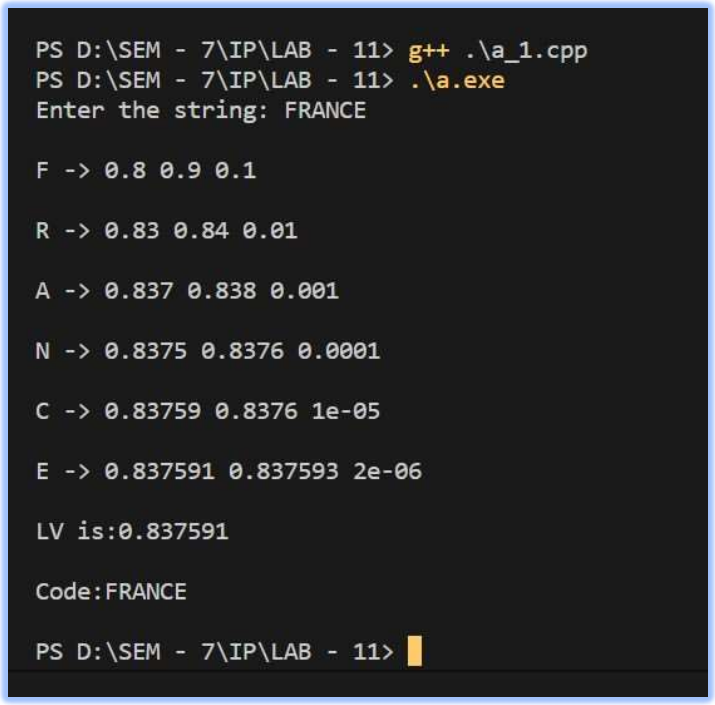


```
    indx[symbol[0]] = 0;
    for (int i = 1; i < symbol.size(); i++)
    {
        indx[symbol[i]] = i;
        if (i > 0)
        {
            rangefrom.push_back(rangeto[i - 1]);
            rangeto.push_back(rangefrom[i] + probability[i]);
        }
    }

    string s;
    cout << "Enter the string: ";
    cin >> s;

    double LV_OLD = 0, HV = 1, DIFF = 1, LV;
    for (int i = 0; i < s.size(); i++)
    {
        LV = LV_OLD + DIFF * rangefrom[indx[s[i]]];
        HV = LV_OLD + DIFF * rangeto[indx[s[i]]];
        DIFF = HV - LV;
        LV_OLD = LV;
        cout << endl
             << s[i] << " -> " << LV << " " << HV << " " << DIFF <<
endl;
    }
    cout << "\nLV is:" << LV << endl;

    // Decoding Arithmetic code
    double code = LV;
    int i;
    int len = s.size();
    string res = "";
    while (len != 0)
    {
        for (i = 0; i < symbol.size(); i++)
        {
            if (rangeto[indx[symbol[i]]] > code &&
rangefrom[indx[symbol[i]]] <= code)
            {
                break;
            }
        }
        res += symbol[i];
        code = (code - rangefrom[indx[symbol[i]]]) /
(rangeto[indx[symbol[i]]] - rangefrom[indx[symbol[i]]]);
        len--;
    }
    cout << "\nCode:" << res << endl
         << endl;
}
```

- **Output:**



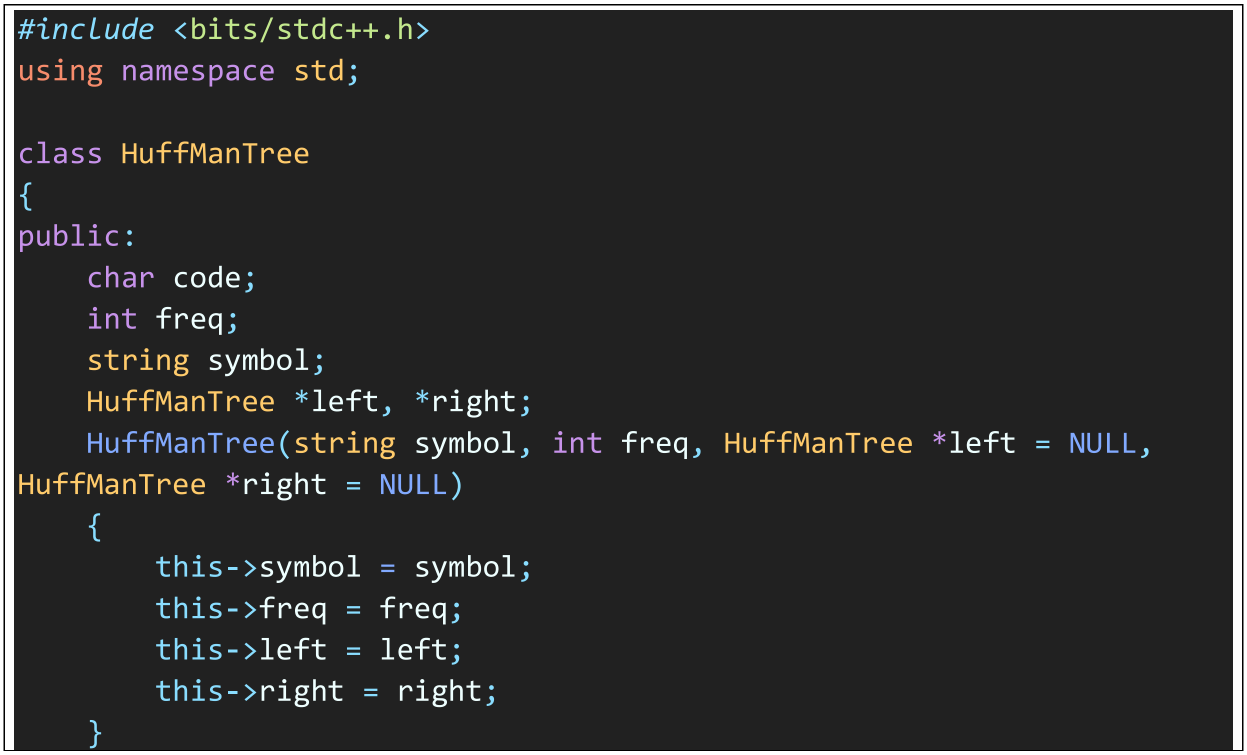
```
PS D:\SEM - 7\IP\LAB - 11> g++ .\a_1.cpp
PS D:\SEM - 7\IP\LAB - 11> .\a.exe
Enter the string: GERMAN

G -> 0.4 0.5 0.1

E -> 0.41 0.43 0.02

R -> 0.416 0.418 0.002

M -> 0.4172 0.4174 0.0002

A -> 0.41734 0.41736 2e-05

N -> 0.41735 0.417352 2e-06

LV is:0.41735

Code:GERMAN

PS D:\SEM - 7\IP\LAB - 11>
```



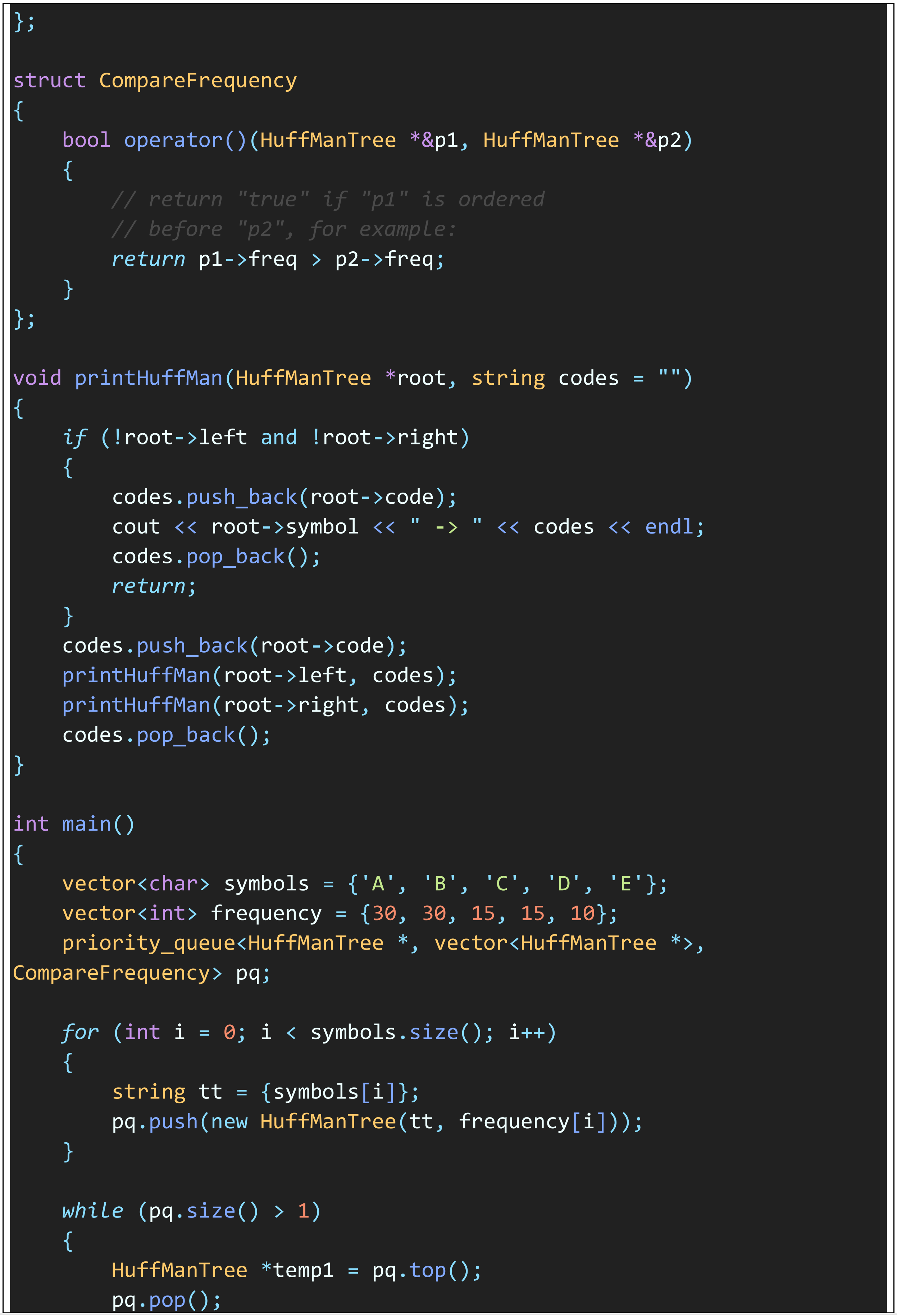
```
PS D:\SEM - 7\IP\LAB - 11> g++ .\a_1.cpp
PS D:\SEM - 7\IP\LAB - 11> .\a.exe
Enter the string: FRANCE

F -> 0.8 0.9 0.1

R -> 0.83 0.84 0.01

A -> 0.837 0.838 0.001

N -> 0.8375 0.8376 0.0001

C -> 0.83759 0.8376 1e-05

E -> 0.837591 0.837593 2e-06

LV is:0.837591

Code:FRANCE

PS D:\SEM - 7\IP\LAB - 11> 
```

Q. 2: Implement Huffman Coding and Decoding.

❖ Code:

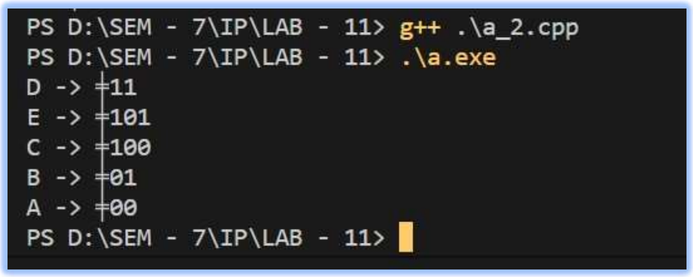


```
#include <bits/stdc++.h>
using namespace std;

class HuffManTree
{
public:
    char code;
    int freq;
    string symbol;
    HuffManTree *left, *right;
    HuffManTree(string symbol, int freq, HuffManTree *left = NULL,
HuffManTree *right = NULL)
    {
        this->symbol = symbol;
        this->freq = freq;
        this->left = left;
        this->right = right;
    }
```



```
};

struct CompareFrequency
{
    bool operator()(HuffManTree *&p1, HuffManTree *&p2)
    {
        // return "true" if "p1" is ordered
        // before "p2", for example:
        return p1->freq > p2->freq;
    }
};

void printHuffMan(HuffManTree *root, string codes = "")
{
    if (!root->left and !root->right)
    {
        codes.push_back(root->code);
        cout << root->symbol << " -> " << codes << endl;
        codes.pop_back();
        return;
    }
    codes.push_back(root->code);
    printHuffMan(root->left, codes);
    printHuffMan(root->right, codes);
    codes.pop_back();
}

int main()
{
    vector<char> symbols = {'A', 'B', 'C', 'D', 'E'};
    vector<int> frequency = {30, 30, 15, 15, 10};
    priority_queue<HuffManTree *, vector<HuffManTree *>,
CompareFrequency> pq;

    for (int i = 0; i < symbols.size(); i++)
    {
        string tt = {symbols[i]};
        pq.push(new HuffManTree(tt, frequency[i]));
    }

    while (pq.size() > 1)
    {
        HuffManTree *temp1 = pq.top();
        pq.pop();
        temp1->code = '1';
        HuffManTree *temp2 = pq.top();
        pq.pop();
        temp2->code = '0';
        pq.push(new HuffManTree(temp1->symbol + temp2->symbol, temp1-
>freq + temp2->freq, temp1, temp2));
    }
    HuffManTree *root = pq.top();
    printHuffMan(root);
}
```

❖ **Output:**



```
PS D:\SEM - 7\IP\LAB - 11> g++ .\a_2.cpp
PS D:\SEM - 7\IP\LAB - 11> .\a.exe
D -> ‡11
E -> ‡101
C -> ‡100
B -> ‡01
A -> ‡00
PS D:\SEM - 7\IP\LAB - 11>
```