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

using namespace std;

//Constructor
HuffmanCode::HuffmanCode(string source){
    data = source;
    heap = new HuffmanHeap(data.length());  //Creates a heap of the size equal to total number of    ↙
    characters.
    buildMap();                             //Build the map<character, frequency>
    buildHeap();                            //Heapification based on the frequency
}

//Destructor
HuffmanCode::~HuffmanCode(){
    delete heap;
}

// Builds the map<character, frequency>. Where the frequency counts the number
// of occurence of the character in the source string.
void HuffmanCode::buildMap(){

    for(size_t i = 0; i < data.length(); i++){
        char c = data.at(i);
        if (frequencyTable.find(c) == frequencyTable.end())     //Is it the first time, the character is   ↙
    witnessed.
        {
            frequencyTable.insert(pair<char, int>(c, 1));       //if yes: Add it to the map and set its   ↙
    frequncy to 1.
        }
        else{
            frequencyTable[c]++;                                //else: increase its frequency count by 1.
        }
    }
}

//Buid Heap
void HuffmanCode::buildHeap(){

    for(map<char,int>::iterator it = frequencyTable.begin(); it != frequencyTable.end(); ++it){     //     ↙
    iterate through the map
            heap->insert(new HuffmanNode(it->first, it->second));                                   //     ↙
    create a huffman leaf node for each character
    //and put them in the heap
    }

    while(heap->getHeapSize() > 1){                             // As long as heap has more than 'one' node ↙
     (the remaining one node is the root node)
        HuffmanNode* left  = heap->removeMin();                 // remove two minimum nodes at a time
        HuffmanNode* right = heap->removeMin();                 // set them as left and as right node
        heap->insert(new HuffmanNode(left, right));             // combine the two and create a single     ↙
    parent node
    }

    string code = "";
    getHuffmanEncoding(heap->getRoot(), code);
    //encode();
}

void HuffmanCode::getHuffmanEncoding(HuffmanNode* root, string code){
    if(root->getLeft() == NULL){                    //Every node has been encoded
        root->setHuffmanCode(code);                 //nothing left to encode
        huffmanTrie.insert(pair<char, string>(root->getLetter(), code)); // leaf node has been coded and   ↙
    insterted in huffmanTrie
        return;
    }
    else{                                                   //Recursively call getHuffmanEncoding again
        getHuffmanEncoding(root->getLeft(), code+"0");      //Appened '0' to the code of the left node
        getHuffmanEncoding(root->getRight(), code+"1");     //Appened '1' to the code of the right node
    }
}


void HuffmanCode::printHuffmanTrie(){
    cout<<"||========== Printing Huffman Trie ==========|| \n"<<endl;
    cout<<"Character    Trie Code\n"<<endl;
    for(map<char,string>::iterator it = huffmanTrie.begin(); it != huffmanTrie.end(); ++it){
        if(it->first == '\n'){cout << "\\n" << "\t \t" << it->second << endl; continue;}
            else if(it->first == ' '){cout << "space";}
            cout << it->first << "\t \t" << it->second << endl;     //Print the character, and then its ↙
    huffman code.
    }
}

string* HuffmanCode::printEncoded()
{
    this->encodedData = "";
    for(int i = 0; i < data.length(); i++)
    {
        encodedData.append(huffmanTrie[data.at(i)]);
    }


    return &encodedData;
}
```