Huffman coding.

Compression (encode):

1) Read data from an input.
2) Store in array frequency of signs (what kind of data type we should to use to store one character - char, string?). You should considerate how to store two data: "frequency" (how many times particular sign occur in input data) and corresponding with that frequency "character" (a sign).
3) Create a Huffman tree (needed to encode data - to replace old sequence of bits (characters from the input) by a new (usually short one - prefix codeword). Huffman tree has to correspond to number of occurrences of character.
4) Create an array of codewords. It can by convenient to create data structure (for example - an array) in order to translate in easy way character from the input to prefix codeword.
5) Write (what kind of writing to the file: binary or text?) the Huffman tree as a stream of bits.
6) Write the number of input characters (encoded as a stream of bits. Decide how many bytes you wont to use to store number of characters; for example it can by 4 or 5 bytes).
7) Write prefix-free codewords (use an array (see point 4) to translate characters from the input to prefix-free codewords).

Decode (unpacking):

1) Read Huffman tree (encoded at beginning of a file).
2) Read a number of characters to decode.
3) Use the Huffman tree to decode remained (compressed) data from a file.

Example 1 (C++). Implementation of Huffman coding algorithm (realization of two first points (blue lines) and partially third point (green lines)). Fields marked with gray background in the following code can be omitted. File with sample text (which can be used to encode) can be download from web page (file: shesells.txt).

[huff_v01.cpp]

```
// huff_v01.cpp
// demonstrates Huffman code (only a few first points from list above are
// completed)
// Andrzej Pisarski
#include <iostream>
#include <fstream>
#include <vector>
#include <string>

using std::vector;
using std::string;
using std::cin;
using std::cout;
using std::endl;
using std::ifstream;
using std::noskipws;
```

```
class Character
{
    public:
        char c;
        int nTimes;

        Character(char ct=-1,int nt=0):c(ct),nTimes(nt)
        {}

        void insChar(char ic){
            c=ic;
        }

        void insnTimes(int it){
            nTimes=it;
        }

        void plusnTimes(){
            nTimes++;
        }

        char getChar(){
            return c;
        }

        int getnTimes(){
            return nTimes;
        }
};

class Link{
    public:
        char LChar;
        int LnTimes;
        Link *pNext;
        Link *pLeftChild;
        Link *pRightChild;

        Link(char lc, int lnt): LChar(lc), LnTimes(lnt), pNext(NULL),
            pLeftChild(NULL), pRightChild(NULL)
        {}

        void display(){
            cout<<LChar<<" "<<LnTimes<<endl;
        }
};

class ReadFile{
    public:
        vector<char> text;
        string fullpath;
        vector<Character> freq;
        Link* pFirst;
        int nElems;
        ReadFile(const string &name, const string &path=""):
            fullpath("  "), pFirst(NULL)
        {
            fullpath=path+name+".txt";
            ifstream f(fullpath.c_str());
```

```
    char temp;
    while(f>>noskipws>>temp)
        text.push_back(temp);

    f.close();

    freq.resize(256);
    for(unsigned int i=0;i<255;i++){
        freq[i].c=i;
    }
}


void display(){
    for(unsigned int i=0;i<text.size();i++)
        cout<<text[i];
}

void findFreq(){
    for(unsigned int i=0;i<text.size();i++){
        freq[text[i]].nTimes++;
    }

}

void displayF(){
    for(unsigned int i=0;i<256;i++){
        int ntemp=freq[i].nTimes;
        if(ntemp>0){
            cout<<freq[i].c<<" "<<ntemp<<endl;
        }
    }
}

void bubbleSort(){
    for(unsigned int i=0;i<freq.size();i++){
        for(unsigned int j=1;j<freq.size()-i;j++){
            if(freq[j].nTimes<freq[j-1].nTimes){
                Character temp=freq[j-1];
                freq[j-1]=freq[j];
                freq[j]=temp;
            }
        }
    }
}

void makeList(){
    if(pFirst==NULL){
        pFirst=new Link(freq[0].c,freq[0].nTimes);

        Link* pPrevious=pFirst;
        for(int i=1;i<256;i++){
            if(freq[i].nTimes>0){
                Link* temp=new Link(freq[i].c,freq[i].nTimes);
                pPrevious->pNext=temp;
                pPrevious=temp;
                nElems++;
            }
        }
    }
}
```

```
        void displayList(){
                Link* pPrevious=pFirst;
                while(pPrevious!=NULL){
                    pPrevious->display();
                    pPrevious=pPrevious->pNext;
                }
        }
};

int main(){

    string filename="shesells";
    ReadFile poem(filename);

    poem.display();

    poem.findFreq();
    poem.displayF();

    cout<<"=============="<<endl;

    poem.bubbleSort();
    poem.displayF();

    cout<<"##############"<<endl;

    poem.makeList();
    poem.displayList();

}
```

Based on:
Robert Lafore – "Teach Yourself Data Structures And Algorithms In 24 Hours"
Robert Sedgewick – "Algorithms".