DTS Lab 9  
huffman compression

Contents

[Objective 2](#_Toc358034740)

[Prototypes 2](#_Toc358034741)

[Desired Output 3](#_Toc358034742)

[Submission 5](#_Toc358034743)

# Objective

Today you will write a series of functions used to compress a file using Huffman’s algorithm. Place all your code in a file called *HuffmanUtils.h*.

**This header file contains only standalone functions. Do not make a class or any global variables.**

# Prototypes

/////////////////////////////////////////////////////////////////////////////  
// Function : generateFrequencyTable  
// Parameters : filePath - the path to the file to open  
// Return : unsigned int\* - a dynamically allocated frequency table   
// Notes : Dynamically allocates an array of 257 unsigned ints representing  
// the count of each character in the file (the index in the array  
// is the char's value). [256] is the total count  
/////////////////////////////////////////////////////////////////////////////  
unsigned int\* generateFrequencyTable(const char\* filePath)

/////////////////////////////////////////////////////////////////////////////  
// Function : generateLeafList  
// Parameters : freqTable - the frequency table to be used to generate leaf   
// nodes  
// Return : vector<HuffNode\*> - a vector containing the leaf nodes  
/////////////////////////////////////////////////////////////////////////////  
vector<HuffNode\*> generateLeafList(unsigned int\* freqTable)

/////////////////////////////////////////////////////////////////////////////  
// Function : generateHuffmanTree  
// Parameters : leafList - the leaf nodes that will appear in our huffman   
// tree  
// Return : HuffNode\* - the root of the generated tree (it will be the top  
// of the queue)  
/////////////////////////////////////////////////////////////////////////////  
HuffNode\* generateHuffmanTree(vector<HuffNode\*>& leafList)

/////////////////////////////////////////////////////////////////////////////  
// Function : generateEncodingTable  
// Parameters : leafList - a vector containing all the leaves in the tree  
// Return : vector<int>\* - a dynamically-allocated array of 256 vectors  
// Notes : each index in the encoding table corresponds to an index in the   
// frequency table  
/////////////////////////////////////////////////////////////////////////////  
vector<int>\* generateEncodingTable(vector<HuffNode\*>& leafList)

/////////////////////////////////////////////////////////////////////////////  
// Function : writeHuffmanFile  
// Parameters : inputFilePath - the path of the file to open for input  
// outputFilePath - the path of the file to open for output  
// freqTable - the frequency table  
// encodingTable - the encoding table  
// Notes : open the input file using ifstream, read characters one at a time,  
// write the encoding path for that character to BitOStream  
/////////////////////////////////////////////////////////////////////////////  
void writeHuffmanFile(const char\* inputFilePath, const char\* outputFilePath,   
 unsigned int\* freqTable, vector<int>\* encodingTable)

/////////////////////////////////////////////////////////////////////////////  
// Function : cleanup  
// Notes : delete each array, and write recursive helper function to delete  
// the tree in post-order  
/////////////////////////////////////////////////////////////////////////////  
void cleanup(unsigned int\* freqTable, HuffNode\* huffTree, vector<int>\* encodingTable)  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : decodeHuffmanFile  
// Parameters : inputFilePath - the file to open and decode  
// outputFilePath - the decoded mesasge, written to a file  
// Notes: Must re-generate the leaflist, tree, and encoding table using  
// the other functions. The frequency table is pulled from the file  
// via the header chunk.  
/////////////////////////////////////////////////////////////////////////////  
void decodeHuffmanFile(const char\* inputFilePath, const char\* outputFilePath)

# Desired Output

Compile and run your code with the DTSLab9.cpp file provided via FSO. Your console output should match the following block identically:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*\* LAB 9: \*\*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* TEST 1 \*\*\*  
Get a Frequency Table from the Huffman Utility for l8input.txt...  
Freq Table  
 (32) : 6  
F(70) : 1  
S(83) : 1  
a(97) : 1  
b(98) : 1  
c(99) : 1  
e(101) : 4  
h(104) : 2  
i(105) : 2  
l(108) : 4  
o(111) : 2  
r(114) : 1  
s(115) : 3  
t(116) : 2  
u(117) : 1  
v(118) : 1

\*\*\* TEST 2a \*\*\*  
Create the list of starting Huffman Nodes (eventual Leaf Nodes)...

\*\*\* TEST 2b \*\*\*  
Now create a Huffman Tree from the Huffman Utility...  
Depth 0 :   
Depth 1 :   
Depth 2 :   
Depth 3 : s e l   
Depth 4 : o t i h   
Depth 5 : r a u c F S b v

\*\*\* TEST 3 \*\*\*  
Next, make an Encoding Table from the Huffman Utility...  
Encoding Table  
Letter : Code : 111  
Letter : F Code : 11000  
Letter : S Code : 11001  
Letter : a Code : 01111  
Letter : b Code : 11010  
Letter : c Code : 10011  
Letter : e Code : 001  
Letter : h Code : 1011  
Letter : i Code : 1010  
Letter : l Code : 010  
Letter : o Code : 0110  
Letter : r Code : 01110  
Letter : s Code : 000  
Letter : t Code : 1000  
Letter : u Code : 10010  
Letter : v Code : 11011  
  
\*\*\* TEST 4 \*\*\*  
Finally, write a complete Huffman File...  
Final Huffman Encoded Data :  
11000100 10010010 11111001 01111101 00101111 01000011 11000101 10011111   
10100010 00100011 10001001 11011011 00110010 11100111 01100101 11000000

Huffman Uncompressed Data Size : 33  
Huffman Compressed Data Size : 16  
  
  
\*\*\* TEST 5 \*\*\*  
decode a Huffman File...  
the quick brown fox jumps over the lazy dog.

# Submission

To submit the lab assignment:

1. Clean, build, and run DTSLab9.cpp with your HuffmanUtils.h and BitStream.h files in Visual Studio (debug mode).
   1. clear up any warnings you encounter.
   2. verify that your output is correct by comparing it to the lab document's Desired Output section, line-by-line.
   3. ensure there are no memory leaks.
2. On your desktop, create a new folder with your name in the following format:
   1. your last name
   2. a comma
   3. a single space
   4. your first name  
      **\* Appropriate capitalization for proper names should be used.**  
      Suitable examples include : "Pollack, Joey"; "De La Paz, Christhian"; "Tjarks, Matthew".
3. Copy your 'HuffmanUtils.h' and 'BitStream.h' files into the folder that you created in step 2. I need both of these files to grade the lab. These are the only files I need and should therefore be the only files you submit.
4. Right-click on the folder and select 'send to->compressed (zipped) folder'.
5. Submit the compressed folder via FSO.