CS 233 COMPUTER SCIENCE II C++

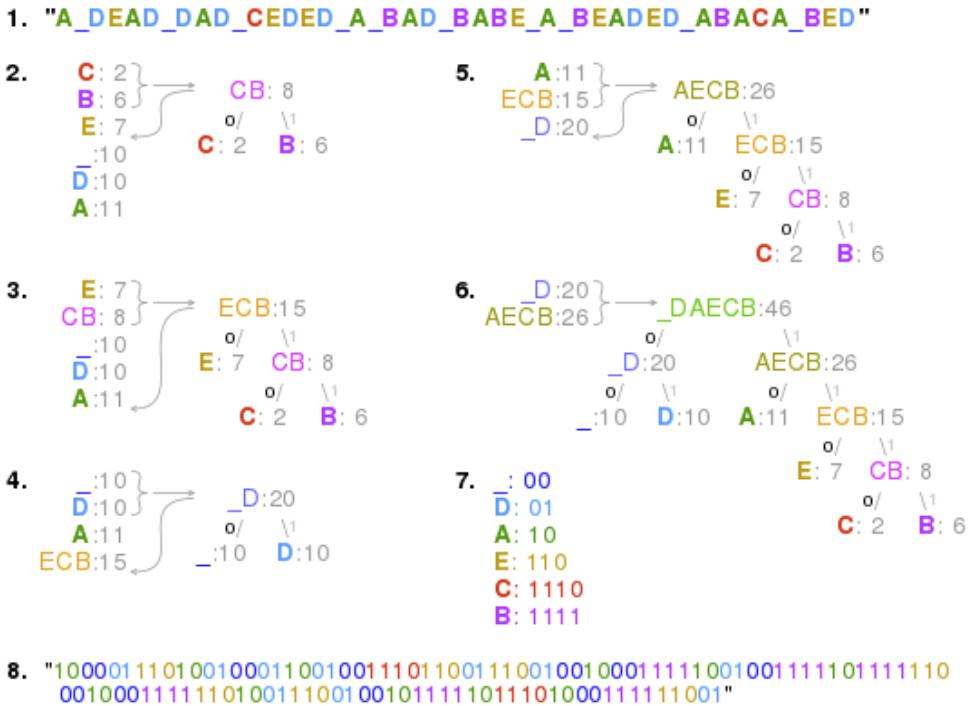
HUFMAN CODING

YES YOU NEED TO FINISH THIS FOR HOME WORK

HUFFMAN CODE

- In computer science and information theory, a Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression.
- The process of finding and/or using such a code proceeds by means of Huffman coding, an algorithm developed by David A. Huffman while he was a Sc.D. student at MIT, and published in the 1952 paper “A Method for the Construction of Minimum-Redundancy Codes”

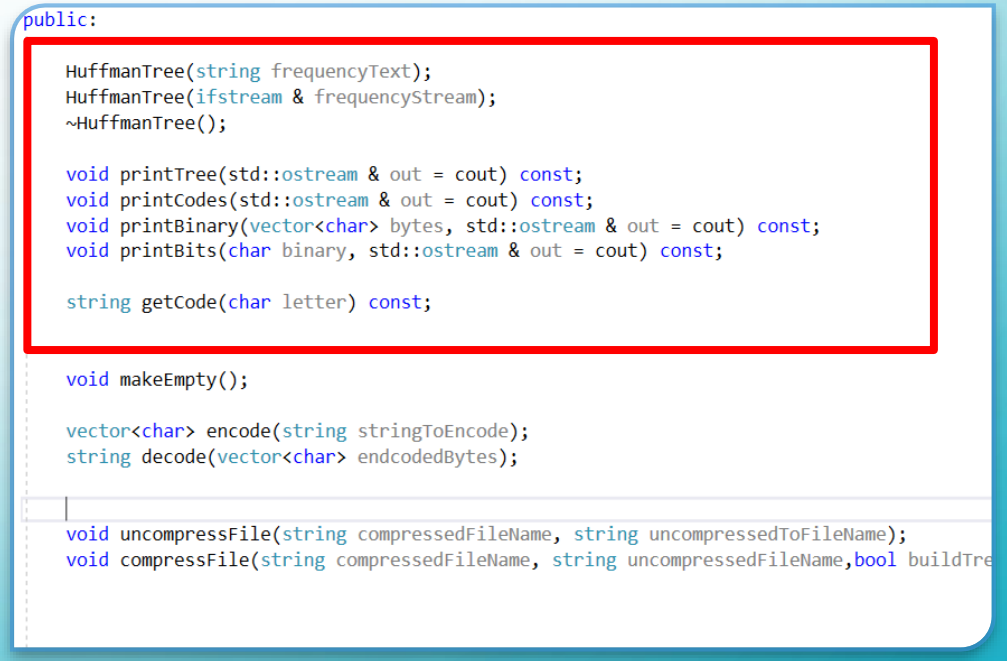
VISUALIZATION



- Visualization of the use of Huffman coding to encode the message "A_DEAD_DAD_CEDED_A_BAD_BABE_A_BEADED_ABACA_BED".
- In steps 2 to 6, the letters are sorted by increasing frequency, and the least frequent two at each step are combined and reinserted into the list, and a partial tree is constructed.
- The final tree in step 6 is traversed to generate the dictionary in step 7.
- Step 8 uses it to encode the message.

YOU STARTED THIS PROJECT IN LAB IN LAB

- For this lab you must complete highlighted methods



```
public:
    HuffmanTree(string frequencyText);
    HuffmanTree(ifstream & frequencyStream);
    ~HuffmanTree();

    void printTree(std::ostream & out = cout) const;
    void printCodes(std::ostream & out = cout) const;
    void printBinary(vector<char> bytes, std::ostream & out = cout) const;
    void printBits(char binary, std::ostream & out = cout) const;

    string getCode(char letter) const;

    void makeEmpty();

    vector<char> encode(string stringToEncode);
    string decode(vector<char> endcodedBytes);

    void uncompressFile(string compressedFileName, string uncompressedToFileName);
    void compressFile(string compressedFileName, string uncompressedFileName,bool buildTre
```

INITIAL TESTING

```
//Test 1
HuffmanTree tree("HHHHEELLLLLLOO WOOOOORRLLLLLLDP");
tree.printTree();
tree.printCodes();
cout << "Code L :" <<tree.getCode('L') << endl;
```

- Start small when testing your lab
- You should also consider getting information as you progress (such as the frequency information)

AFTER THIS TRY READING SOME OF THE FILES

```
//Test 3
std::ifstream frequencyStream("20000leagues.txt");
HuffmanTree tree2(frequencyStream);
tree2.printTree();
tree2.printCodes();
```

```
std::ifstream frequencyStream("Bigo.txt");
HuffmanTree tree2(frequencyStream);
tree2.printTree();
tree2.printCodes();
```

- After your earlier success try larger files

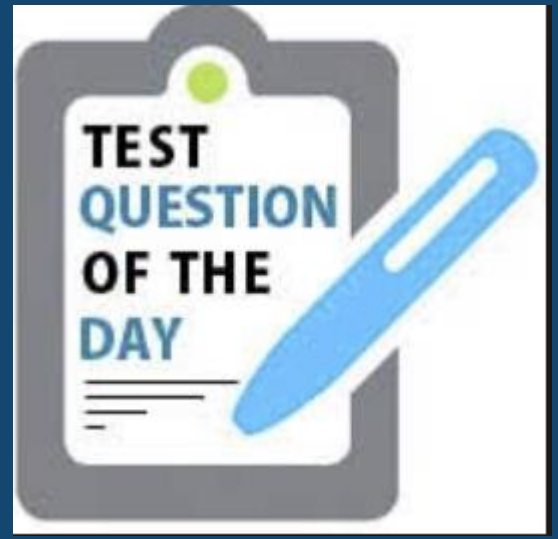
INLINE HINT: TEST QUESTION

- In the C and C++ programming languages, an inline function is one qualified with the keyword inline; this serves two purposes.

```
inline unsigned char HuffmanTree::setBit(unsigned char byte, int position) const
{
    return  byte | BITMASK[position];
}
```

INLINE HINT: THIS WILL BE A TEST QUESTION

- Firstly, it serves as a compiler directive that suggests (but does not require) that the compiler substitute the body of the function inline by performing inline expansion, i.e. by inserting the function code at the address of each function call, thereby saving the overhead of a function call.



INLINE
HINT: NOT A TEST QUESTION

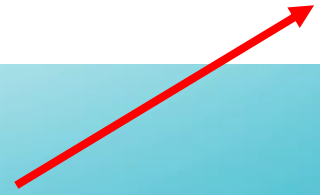
- The second purpose of inline is to change linkage behavior; the details of this are complicated.
- This is necessary due to the C/C++ separate compilation + linkage model, specifically because the definition (body) of the function must be duplicated in all translation units where it is used, to allow inlining during compiling, which, if the function has external linkage, causes a collision during linking (it violates uniqueness of external symbols).
- C and C++ (and dialects such as GNU C and Visual C++) resolve this in different ways

HINTS

- This code will open a file for binary input.

  Note: compressedFileName is a string

```
ifstream compressedFile(compressedFileName, ios::out | ios::binary);
```



Notice the use of a bitwise or!
This allows a single byte to represent several states at once

HINTS

- This code will read a file into a vector of characters

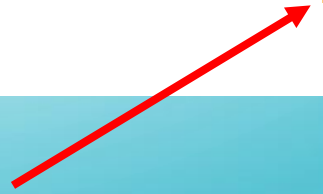
```
ostringstream ss;
ss << compressedFile.rdbuf();
const string& s = ss.str();

vector<char> vec(s.begin(), s.end());
```

HINTS

- This code will open a file for binary output.

  Note: compressedToFileName is a string

```
ofstream compressedFile(compressToFileName, ios::out | ios::binary);
```



Notice the use of a bitwise or!
This allows a single byte to represent several states at once

HINTS

- This code will write a vector of character to a file.

```
vector<char> encoded = encode(compressText);
compressedFile.write(reinterpret_cast<const char*>(&encoded[0]), encoded.size() * sizeof(char));
```

HINTS

- Opening a file to read text

```
ifstream uncompressedFile(uncompressedFileName);
```

HINTS

- Opening a file to write text

```
ofstream uncompressedFile(uncompressedToFileName);
```