**Project 2**

**Data Structures 303**

**Caleb Iamkui**

**Matt Sunderland**

**Daniel Coulson**

**Table of Contents**

**Tables**........................................................................................................................................................3-4

**Algorithm**…………………………………………………………………………………………………………………………………………….5-7

**UML Diagram**………………………………………………………………………………………………………………………………………….8

**Assumptions**……………………………………………………………………………………………………………………………………………9

**Decisions**……………………………………………………………………………………………………………………………………………9-10

**Appendix**………………………………………………………………………………………………………………………..……**End of report**

**BTree Class**

Functions

|  |  |
| --- | --- |
| BTree() | Constructor |
| ~BTree() | Destructor |
| Void insert(char theletter, string thecode, int length) | Insertion called in main to call recursive insert |
| Void encode(string message) | Passes a string to be encoded |
| Void decode(string message) | Passes a string to be decoded |
| Void Insert(Node \*ptr, int count, char theletter, string thecode, int length) | Recursive insertion function |
| String search\_letter(char letter, Node \*ptr, bool &ischar) | Search for a letter and returns a string of the path to that letter |
| Void destroy\_tree(Node \*ptr) | Destroys the tree |
| Node(char theletter, Node \*left\_link, Node \*right\_link) | Gives attributes variables |

Attributes

|  |  |
| --- | --- |
| Char letter | Variable for the characters |
| Node \*left | Variable for left nodes |
| Node \*right | Variable for right nodes |
| Node \*root | Variable for the root |

**Data Class**

Functions

|  |  |
| --- | --- |
| Void PriorityQueueResults(Item\_TypeA& theClassArray, Item\_TypeB& priorityqueue) | Template<class Item\_Type>  Builds tree in a priority queue |
| Void readInFile(ifstream& fin, Item\_Type& addy) | Template<class Item\_Type>  Reads in the text file |
| Void file\_error(ifstream& fin) | Tells user file not found drastically |
| Void cleanup(ifstream&, fin) | Closed file goes here |
| Void printState(Item\_Type& theClassArray) | Prints text file on screen |
| String getcode() | Returns the code |
| Char getletter() | Returns the letter |
| Int getlength() | Returns the length |

Attributes

|  |  |
| --- | --- |
| Char letter | Variable for the characters |
| String code | Variable for the string (message) |
| Int length | Variable for the length of string |

**Main**

Functions

|  |  |
| --- | --- |
| Void encode\_message(BTree &morseTree) | Will pass the encode function into main, making main look more clean |
| Void decode\_message(BTree &morseTree) | Will pass the decode function into main, making main look more clean |

**Algorithm**

1. This algorithm will build a tree out of a text file with the letters and symbols for Morse code. When the text file is passed into main, this class will be called into main to help build the text from the file into a class array.
2. The program will then run a function deciding whether the file was read in correctly.
3. If the debug is read in true, the array will be shown on screen to show the letters and symbols for the Morse code.
4. If the debug is read in false, it will continue and not print the letters and symbols for the Morse code
5. Main will pass the text from the file and insert the text into a priority queue to pre-sort the data and then build the tree through a vector.
6. The binary tree should match the one from the book on page 509. The user will then be prompt to input a message to the screen.
7. The main will then pass either the decode prompt or the encode prompt for the users message.

**Pseudo-Code**

Declare a class to build the binarytree:

class BTree{  
private:  
struct Node{  
char letter;  
Node \*left;  
Node \*right;  
Node(char theletter, Node \*left\_link, Node\* right\_link)  
: letter(theletter), left(right\_link), right(right\_link) {}  
Node \*root;

};  
//recursive insertion function.  
void Insert(declares the pointer, the letter, the code, and the lengt);   
//recursive function to search for letter and return string of the path to that letter  
string search\_letter(declares the letter, the pointer, and true or false on the character); void destroy\_tree(declares the pointer);//recursive function called by destructor to destroy the tree  
public:  
BTree();//default constructor. sets first root.  
~BTree();//destructor  
void insert(passes the letter, the code, and the length);//insertion called in main to call recursive insert  
void encode(string message);//main passes a string to be encoded to this function  
void decode(string message);//main passes a string to be decoded to this function  
};

int main()   
{   
bool debug =false; //used to check through program if true it will print extra data items   
   
BTree morseTree; //declare class Tree   
Data morseData; //declare class Data   
declare an array of 26 elements representing letters of alphabet, or as stated later   
   
//read file into MorseCode   
ifstream fileIN ("morse.txt", ios::in); //open morse.txt for reading in   
   
 if( fileIN.is\_open())   
 {   
 reads in file filestream, and add to class array   
 }   
   
 else   
 { close if file does not exist };  
   
 if(debug==true) //both this line and the following can be deleted   
 morseData.printState(MorseCode);//should be presorted current state to confirm data was read in use debug bool flag   
   
 create a priority queue that will insert by height from min to max   
 for (loop through the 26 characters)   
 pushing array of Morse data objects to vector pq   
 send address of priority\_queue and ClassArray and on ending modifies order of ClassArray   
   
 if(debug==true) //additional debug or printout to verify the sort and the structArray is working correctly, this complete block   
 { //can be deleted if desired   
 cout<<"\nValue After Priority Queue\n";   
 morseData.printState(MorseCode);   
 }  
 //test values can be passed from DataArray! Delete this before submitting   
   
 for (loop through the MorseCode)   
{   
put creating tree in function   
}   
 system("pause");   
return 0;   
}  
void encode\_message(BTreee &morseTree)

{

/////// ENCODING /// put this into function   
string user\_message;   
cout<<"Please enter a message to encode(lowercase letters only): ";   
getline(cin, user\_message);   
//now send message to be encoded and result printed out to screen   
 }

Void decode\_message(BTree &morseTree)

{  
/////////// DECODING ////////// put this into function   
cout<<endl<<endl<<endl;   
cout<<"use \* as delimiter between letters and \*\* between words"<<endl;   
cout<<"Please enter message to be decoded: ";   
getline(cin, user\_message);   
send message to be decoded   
}

**UML Diagram**

|  |
| --- |
| Main |
| encode\_message()  decode\_message() |

|  |
| --- |
| BTree |
| Char letter  Node \*left  Node \*right  Node \*root |
| BTree()  ~BTree()  Insert()  insert()  encode()  decode()  search\_letter()  destroy\_letter()  Node() |

|  |
| --- |
| PriorityQueue<Data, vector<Data>, Data::Comp> pq |
| Push() |

**Assumptions**

1. Program must encode and decode a message from the user using Morse code
2. The binary tree needed to be created from the page 509 figure
3. A space (‘ ‘) was not included in the binary tree, so we assumed to create our own
4. Program does not need to loop
5. The input file is formatted correctly with a letter, a space, then followed by the code given
6. The message from the user can ONLY use lowercase letters from the English alphabet
7. User types in the message to be decoded correctly, following the directions given

**Decisions**

Stated Problem: p. 509  
7. Morse code (see Table 8.11) is a common code that is used to encode messages consist-  
ing of letters and digits. Each letter consists of a series of dots and dashes; for example,  
the code for the letter a is ������ and the code for the letter b is ������������ . Store each letter of  
the alphabet in a node of a binary tree of depth 4. The root node is at depth 0 and  
stores no letter. The left node at depth 1 stores the letter e (code is ���) and the right node  
stores the letter t (code is ���). The four nodes at depth 2 store the letters with codes (������,  
������, ������, ������). To build the tree (see Figure 8.38), read a file in which each line consists of a  
letter followed by its code. The letters should be ordered by tree depth. To find the posi-  
tion for a letter in the tree, scan the code and branch left for a dot and branch right for  
a dash. Encode a message by replacing each letter by its code symbol. Then decode the  
message using the Morse code tree. Make sure you use a delimiter symbol between  
coded letters.  
  
(from Lecture/slide 20)  
-Read the letters and their codes from a file. The letters are not sorted in the file.   
You will have to sort your letters so that it is built correctly.  
  
-Build the tree so that it becomes a morse code tree.  
  
-Encode a message (Convert a message from characters into morse code).  
You will be visiting your tree. If you visit a left child, you append ���.   
Otherwise, you append \_  
  
-Decode a message (Convert a message from morse code into letters).

Design Notes  
Stage 1  
Implement (build) a huffman tree given a input file of symbol and code.  
i.e. an unordered list with the first char representing the symbol to represent the node and the   
dot, dash pattern to represent the code ( this will be the traversal path to find where the symbol is stored)  
  
a \* -   
z - - \* \*  
c - \* - \*  
  
Stage 2  
Compose a message using the ASCII character  
- use a search to find the value then traverse to root printing out the path of left/\right   
  
Stage 3  
The message from Stage 2, assumed is in a buffer. Repeat Stage 1 without writing over the existing tree.   
  
/\*\*Brief summary of program  
? Represents possible use but not tested or discussed method to solve.  
  
STAGE 1 (BUILD TREE)  
- READ IN FILE   
? USE char cSymbol and string sCode //stores ascii abc value and string value of dots and dashes  
- INSERT TO HEAP   
?BY ITS NATURE A HEAP SHOULD SORT AS INSERTED CORRECT?  
  
STAGE 2 (SEARCH SYMBOL)  
- SEARCH FUNCTION TO LOCATE NODE WITH SYBMOL  
? return the value of sCode per found node into a buffer  
  
STAGE 3 (TRAVERSE TREE FROM ROOT using sCODE)  
- USE SIMILAR FUNCTION TO STAGE 1, DO NOT CHANGE EXISTING HEAP ONLY RETURN NODE FROM sCODE END NODE POSITION