**C PROJECT**

**Title: Morse Code Converter**

**Point Value: 100**

Samuel F. B. Morse produced the first working telegraph set in 1836. This made transmission possible over any distance. The first Morse Code message, "What hath God wrought?", was sent from Washington to Baltimore.

Morse code was extensively used for early radio communication beginning in the 1890s.

In the early part of the twentieth century, the majority of high-speed international communication

was conducted in Morse code, using telegraph lines, undersea cables, and radio circuits.

Morse code can also be transmitted using light which sometimes happens between ships at sea. It is used in emergencies to transmit distress signals when no other form of communication is available. The standard international distress signal is •••---••• (SOS).

A sheet of music

Description automatically generated with medium confidence

Morse code can easily be converted into English using a binary tree set up in the following manner, where a ‘.’ Moves to the left and ‘-‘ moves to the right.

A picture containing scissors

Description automatically generated

An a is ‘.-‘which moves from the root to the left and then to the right. Each letter will be delimited by a space (‘ ‘). Each word will be delimited by a ‘/’.

This string: “.... . .-.. .-.. --- / .-- --- .-. .-.. -..” is converted into “hello world”

**Building the Binary Tree (MorseTree)**

The nodes of the tree will be a structure which holds the letter and the links (addresses) of the right child and left child.

//node of our tree

struct node {

char letter;

struct node\* left;

struct node\* right;

};

You will need to build the tree in a preOrder fashion. In other words, the node with ‘e’ must be in existence before the nodes for ‘i’ and ‘a’ can be linked as the left and right child of ‘e’. I would suggest including a preOrder traversal function to test your buildMorseTree.

**Using the MorseTree**

Use the MorseTree to convert Morse Code to English by taking the code and finding it’s corresponding English letter by traversing the MorseCode Tree, ‘.’ branches to the left and ‘-‘ branches to the right. The code ‘.--. ’ would branch to the left, then to the right, then to the right, then to the left then a space which indicates to return the letter ‘p’. Each letter is delimited by a space (‘ ‘). Each word is delimited by a ‘/’.

Start with the main.c given to you.

You will be implementing the following functions:

1. struct node\* createNode(char character) {}

dynamically allocate memory for a node and set the values of letter, left and right.

1. void buildMorseTree(struct node\*root) {}

Build the MorseTree to reflect the binary tree above.

1. void printMorseTree(struct node \*root) {}

Prints out the MorseTree in preorder.

1. void convertMorseToEnglish(char\* **characters**, char\* **output**){}

This function takes two parameters – **characters** and **output** which are \*char. **characters** is the morse code characters to convert and **output** is where the English translation will be stored. Note this function doesn’t have a return value, **output** must be created and passed in as a parameter. Since it is passed as a pointer, any manipulation of **output** is permanent. This function is responsible for creating the root and building the MorseTree by calling buildMorseTree and then using the MorseTree to convert the morse code characters into English characters. At the end of the function, place a ‘\0’ at the end of **output**. This function will also call printMorseTree after calling buildMorseTree to show that tree has been built correctly.

1. int main(int arc, char\*\* argv){}. This is the driver function for the convertMorseToEnglish. It will:

* read from a textfile the morse code characters, add a space and \0 at the end
* create the output array
* Call convertMorseToEnglish
* Print out the resulting output

Optional functions you may consider:

1. convertCharacter – converts a string of dots and dashes to a character
2. addRight – adds to the right child of a node
3. addLeft – add to the left child of a node

Place textfiles in the same folder as <yourfilename>.c file.

You will compile and execute as follows on the gl server

gcc -o convert <yourfilename>.c

convert <textfile>.txt

Some suggestions:

1. There is a morse code translator at:

<http://morsecode.scphillips.com/jtranslator.html>

it will help you make test cases

A sample run is provided (file "C Project output.docx" attached to the blackboard assignment) to ensure that you understand the overall operation of the program.

**What to turn in:**

You will need to submit the following files in a zipped folder labeled **lastname\_Cproject.zip** or **lastname\_lastname\_Cproject.zip** if two of you are working together on this project.

1. Set of instructions (algorithm) for your createNode, buildMorseTree and convertMorseToEnglish functions. You can use pseudo-code. Make sure it is clear enough that someone else could read it and understand it.
2. Set of tests – a set of 10 tests for this program (the first one is done for you as an Example).

|  |  |
| --- | --- |
| Morse Code | English |
| .... .- .--. .--. -.-- / -... .. .-. - .... -.. .- -.-- | happy birthday |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

1. 3 .txt files. Each .txt file will represent one of the 10 test inputs.
2. Your updated main.c
3. Screen copy of your program working on the gl server with your 3 .txt files built from 3 of your 10 test cases.

**Implementation Details:**

* You may work in groups of 1 or 2.
* You program must run successfully on the gl server.
* Algorithm, test cases and screen copies can be turned in, in one .docx document
* Code that does not compile on the gl server will receive up to 50%.
* Any late submissions will incur a penalty of 10% points per every portion of 24 hours they are late. If received 15 mins late – 10% penalty. If received 25 hours late – 20% penalty. Nothing will be accepted after 3 days late.

**Grading Breakdown:**

The 100 total points for this part will be broken down into expected functionality (how well the code works), and how well the code is written. The points for each part include the expected error handling.

* [85] Program functionality:
  + [10] createNode
  + [20] buildMorseTree
  + [10] printMorseTree
  + [35] convertMorseToEnglish
  + [10] main, reads from a file, calls convertMorseToEnglish and prints out result
* [15] Pseudocode, test cases, screenshots