Morse Code Translator

Morse code is a rhythmic human comprehensible transmission code. Developed to transmit messages long distances before the invention of the telephone. Morse code is still useful today because it can transmitted over wire, radio, light, or trumpet.

This program uses a binary tree for storing and searching, making it faster. Expressing in terms of big-O notion, a simple array search would take O(n), this only take O(n log n).

Morse Code Translator was written by Devin Watson.

In the margins of page 2 you'll find the listing for the data file you need that defines the Morse code alphabet.

/* morsetran.c listing begins: */

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define DICTIONARY_FILE "morse.txt"
#define true (-1)
#define false (0)
#define maxlen (20) /* Length of strings in console */
#define NUM_COMMANDS 8

char commands[NUM_COMMANDS][maxlen]= {
    "morse","english","morsefile","engfile",
    "help", "info", "quit", "exit"
};

int convertToMorse(char *);
int convertToEnglish(char *);
int morseFile(char *);
int engFile(char *);
int doHelp(char *);
int doQuit(char *);
int doTree(char *);

int (*comfuncs[NUM_COMMANDS])(char *)= {
    convertToMorse, convertToEnglish, morseFile, engFile,
    doHelp, doHelp, doQuit, doQuit
};

/* Remove all c's from string s */
void stripout(char *s,const char c) {
    size_t i = 0, j = 0;
    while(i<strlen(s)) {
        if(s[i]!=c) { s[j]=s[i]; j++;}
        i++;
    }
    s[j]=0;
}

/*
    Split string s at first space, returning first 'word' in
    t & shortening s
*/
void spacesplit(char *s,char *t) {
    size_t i=0,j=0;
    size_t l=strlen(s);
    while((i<l)&(s[i]==' ')) i++;; /* Strip leading spaces */
    if(i==l) {s[0]=0; t[0]=0; return;};
    while((i<l)&(s[i]!=' ')) t[j++]=s[i++];
    t[j]=0;    i++; j=0;
    while(i<l) s[j++]=s[i++];
    s[j]=0;
}

/* Return nonzero if and only if string t begins with non-empty string s */
int stringbeg(char *s,char *t) {
    size_t i=0;
    size_t l=strlen(s);
    if(l>0) {
        while((i<l)&(toupper(s[i])==toupper(t[i]))) i++;
        if(i==l) return true;
```

/* Listing continued on next page…*/

/* Listing continued from previous page */

```
    }
  return false;
}

/* Check string s against n options in string array a
   If matches ith element return i+1 else return 0 */
unsigned short stringmatch(char *s,char a[][20], unsigned short n) {
    unsigned short i=0;
    while(i<n) {
        if(stringbeg(s,a[i])) return i+1;
        i++;
    }
    return 0;
}

/* Execute command s */
int parser(char *s) {
    unsigned short i;
    char c[maxlen];
    spacesplit(s,c);
    i=stringmatch(c,commands,NUM_COMMANDS);
    if(i)return (*comfuncs[i-1])(s) ;
    printf("\n Bad command (");
    printf(c);
    printf(")");
    return false;
}

/*
    The treeNode structure
    holds the morse characters
    and the corresponding English letter
*/
struct sTreeNode {
    char* morse;
    char* letter;
    struct sTreeNode *left, *right;
};

typedef struct sTreeNode treeNode;

treeNode *root;
treeNode *morseTree;

/*
    Frees memory in use by a tree node
*/
void free_node(treeNode *node) {
    /*
        The letter and morse members
        were dynamically allocated,
        so we should free them first
        just to be safe.
    */
    free(node->letter);
    free(node->morse);
    free(node);
}

/*
    Recursive deletion of
    all nodes in tree
*/
```

morse.txt

```
A|.-
B|-...
C|-.-.
D|-..
E|.
F|..-.
G|--.
H|....
I|..
J|.---
K|-.-
L|.-..
M|--
N|-.
O|---
P|.--.
Q|--.-
R|.-.
S|...
T|-
U|..-
V|...-
W|.--
X|-..-
Y|-.--
Z|--..
,|--..--
?|..--..
!|..--.
.|.-.-.-
/|-..-.
@|.--.-.
0|-----
1|.----
2|..---
3|...--
4|....-
5|.....
6|-....
7|--...
8|---..
9|----.
:|---...
"|.-..-.
'|.----.
=|-...-
+|.-.-.
```

/* Listing continued on next page…*/

/* Listing continued from previous page */

```
void deleteTree(treeNode **tree) {
    treeNode* old_node = *tree;

    if ((*tree)->left == NULL) {
        *tree = (*tree)->right;
        free_node(old_node);
    } else if ((*tree)->right == NULL) {
        *tree = (*tree)->left;
        free_node(old_node);
    } else {
        treeNode** pred = &(*tree)->left;
        while ((*pred)->right != NULL) {
            pred = &(*pred)->right;
        }

        deleteTree(pred);
    }
}

/*
    Utility function to remove newline
    characters from the end of a string
*/
void chomp(char *s) {
    s[strcspn(s,"\n")] = '\0';
}

/*
    Quick comparison
*/
int compare(char *char1, char *char2) {
    return strcmp(char1, char2);
}

/*
    Search the tree for Morse code equivalent
    of letter. Returns the found treeNode or NULL
    if not found.
*/
treeNode** findMorse(treeNode** root, char *let) {
    if (let == " ") return NULL;
    treeNode** node = root;
    int compare_result;

    while (*node != NULL) {
        compare_result = compare(let, (*node)->letter);
        if (compare_result < 0)
            node = &(*node)->left;
        else if (compare_result > 0)
            node = &(*node)->right;
        else
            break;
    }
    return node;
}

treeNode** findLetter(treeNode** root, char *morse) {
    if (morse == " ") return NULL;
    treeNode** node = root;
    int compare_result;

    while (*node != NULL) {
        compare_result = compare(morse, (*node)->morse);
```

/* Listing continued on next page…*/

/* Listing continued from previous page */

```
        if (compare_result < 0)
            node = &(*node)->left;
        else if (compare_result > 0)
            node = &(*node)->right;
        else
            break;
    }
    return node;
}

/*
    Converts English to Morse Code
*/
int convertToMorse(char *s) {
    int i = 0;
    char *temp_letter;
    treeNode **tmpNode;

    printf ("Translating %s\n\n",s);
    printf ("Morse:\n");

    while (s[i] != '\0') {
        if (s[i] != ' ') {
            temp_letter = (char *)malloc(sizeof(char)*2);
            temp_letter[0] = (char)toupper(s[i]);
            temp_letter[1] = '\0';
            tmpNode = findMorse(&root,temp_letter);

            if ( (*tmpNode) != NULL) {
                printf("%s ",(*tmpNode)->morse);
            } else {
                printf("$"); //Unknown symbol
            }
            free(temp_letter);
        } else {
            printf(" ");
        }
        i++;
    }
    return -1;
}

/*
    Convert Morse Code to English
    This one is a little trickier
*/
int convertToEnglish(char *s) {
    char *pch;
    treeNode **temp_node;

    printf("Translating %s\n\n", s);
    printf("English:\n");

    pch = strtok (s," ");
    while (pch != NULL) {
        //Fullstop character, print space
        temp_node = findLetter(&morseTree,pch);
        if ( (*temp_node) != NULL) {
            printf(" %s ",(*temp_node)->letter);
        } else {
            printf(" $ "); //Unknown symbol
        }
```

/* Listing continued on next page…*/

/* Listing continued from previous page */

```
        pch = strtok(NULL, " ");
    }
    return true;
}

int morseFile(char *s) {
    return true;
}

int engFile(char *s) {
    return true;
}

int doHelp(char *s) {
    (void)(&s);
    printf("\nCommands available:");
    printf("\nMorse <text>         English to Morse");
    printf("\nEnglish <morse>      Morse to English");
    printf("\nQuit or Exit         Exit Program");
    printf("\nHelp or Info         Display this text");
    printf("\n\nCommands are case-insensitive,\n");
    printf("eg., Morse and morse are the same");

    return true;
}

// Exits from program, after doing cleanup
int doQuit(char *s) {
    (void)(&s);
    deleteTree(&root);
    deleteTree(&morseTree);
    exit(0);
    return(0);
}

/*
    Recursive insert into tree
    Finds the most balanced location
*/
void treeInsert(treeNode ** tree, treeNode * item) {
    // Base case -- Tree is empty,
    // so this becomes the root node
    if(!(*tree)) {
        *tree = item;
        return;
    }

    int result = strcmp(item->letter, (*tree)->letter);
    if (result < 0)
        treeInsert(&(*tree)->left, item);
    else if (result > 0)
        treeInsert(&(*tree)->right, item);
}

void morseTreeInsert(treeNode ** tree, treeNode * item) {
    if(!(*tree)) {
        *tree = item;
        return;
    }

    int result = strcmp(item->morse, (*tree)->morse);
    if (result < 0)
        morseTreeInsert(&(*tree)->left, item);
```

/* Listing continued on next page…*/

/* Listing continued from previous page */

```
    else if (result > 0)
        morseTreeInsert(&(*tree)->right, item);
}

/*
    Load data from dictionary
    file into tree.
    Returns -1 on failure, 1 on success
*/
int loadData() {
    FILE *fp;
    char mystring[100];
    char *let;
    char *morse;
    char *pch;
    treeNode *node;

    fp = fopen(DICTIONARY_FILE,"r+");
    if (fp == NULL) return -1;
    while (!feof(fp)) {
        let = NULL;
        morse = NULL;
        fgets(mystring,100,fp);

        pch = strtok(mystring,"|");
        let = pch;
        morse = strtok(NULL,"|");
        chomp(morse);

        node = (treeNode *)malloc(sizeof(treeNode));
        node->left = node->right = NULL;

        node->letter = (char *)malloc(sizeof(char)*strlen(let));
        strcpy(node->letter,let);
        node->morse = (char *)malloc(sizeof(char)*strlen(morse));
        strcpy(node->morse,morse);
        treeInsert(&root,node);
        morseTreeInsert(&morseTree,node);
    }
    fclose(fp);
    return 1;
}

int main(int argc, char *argv[]) {
    char getcommand[maxlen];
    //Initialize tree
    root = NULL;
    morseTree = NULL;
    // Load dictionary into tree
    if (loadData() == -1) {
        printf("Error loading dictionary file %s, aborting", DICTIONARY_FILE);
        return 1;
    }

    printf("\nMorseTran\n");
    doHelp("");
    for(;;) {
        printf("\n\nCommand? ");
        gets(getcommand);
        parser(getcommand);
    }

    return 0;
}
```

/* Listing continued from previous page */

/* Listing continued on next page…*/