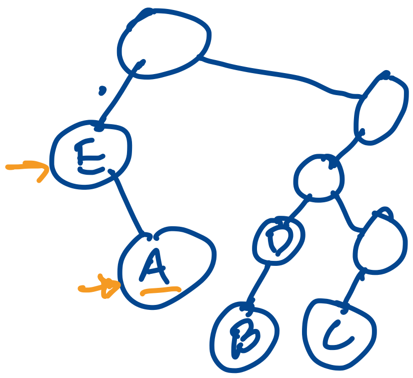
11.13. Homework 13 - Morse Code Decoder

This assignment uses a tree data structure to decode a file encoded with the Morse code (dots and dashes). My colleague, Professor Mertz, wrote an introduction to how Morse code works and how a tree data structure can be used to decode the sequence of dots and dashes to the letters that they represent. `morsecode.pdf`.

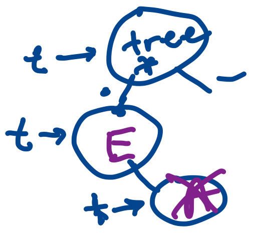
Here is a message for you to decode. In this file, there is a space between each letter and three space characters between each word. `morse_encoded.txt`

Below is most of the program. All of the *main* is function is provided to build the tree, read the file, parse the input into letters, call the decoding function, and display the output. You need to write the short function to traverse the tree and return each letter.

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

#define MAXLINE 150
#define true 1
#define false 0

typedef struct mtree {
    char letter;
    struct mtree *dash;
    struct mtree *dot;
} Morsetree;

int ReadLine(char *, int, FILE *);
void add_morse(Morsetree *, char *, char);
Morsetree *morsetree(char);
char read_morse(Morsetree *, char *);
char *multi_tok(char *, char *);

int main(void)
{
    Morsetree *r;
    char filename[50], line[MAXLINE];
    FILE *fp;
    char *word, *q1, *letter, *q2;

    /*
     * First, build the tree
     */
    r = morsetree('*');
    add_morse(r, ".-", 'A');
    add_morse(r, "-..." ,'B' );
    add_morse(r, "-.-.", 'C' );
    add_morse(r, "-..", 'D');
    add_morse(r, ".", 'E');
    add_morse(r, "..-.", 'F');
    add_morse(r, "--.", 'G');
    add_morse(r, "....", 'H');
    add_morse(r, "..", 'I');
```



```
    add_morse(r, ".---", 'J');
    add_morse(r, "-.-", 'K');
    add_morse(r, ".-..", 'L');
    add_morse(r, "--", 'M');
    add_morse(r, "-.", 'N');
    add_morse(r, "---", 'O');
    add_morse(r, ".--.", 'P');
    add_morse(r, "--.-", 'Q');
    add_morse(r, ".-.", 'R');
    add_morse(r, "...", 'S');
    add_morse(r, "-", 'T');
    add_morse(r, "..-", 'U');
    add_morse(r, "...-", 'V');
    add_morse(r, ".--", 'W');
    add_morse(r, "-..-", 'X');
    add_morse(r, "-.--", 'Y');
    add_morse(r, "--..", 'Z');

    /*
     * Read Morse encoded file, parse it into words, and then decode it.
     */
    printf("Enter the filename of Morse encoded file: ");
    scanf("%s", filename);
    printf("\n");
    if ((fp = fopen(filename, "r")) == NULL) {
        printf("cannot open file %s\n", filename);
        return 1;
    }
    while( ReadLine(line, MAXLINE, fp) ) {
        // split line into words
        // - strtok won't work because of multiple spaces in the token
        for( q1 = line; (word = multi_tok(q1, "  ")) != NULL; q1 = NULL ) {
            // split the word into letters
            for( q2 = word; (letter=strtok(q2, " ")) != NULL; q2 = NULL ) {
                // decode the Morse code letter and display it
                putchar( read_morse(r, letter) );
            }
            putchar( 32 ); // ascii space after each word
        }
        putchar( 10 ); // ascii line feed after each line
    }
    fflush(stdout);
    fclose(fp);
    return 0;
}

Morsetree *morsetree(char l)
{
    Morsetree *new;

    if( (new = (Morsetree *)malloc(sizeof(Morsetree))) == NULL) {
        fprintf( stderr, "Memory Allocation error.\n" );
        return new;
    }
    new->letter = l;
    new->dot = NULL;
    new->dash = NULL;
    return new;
}

void add_morse(Morsetree *tree, char *morse, char letter)
{
    Morsetree *t;
    int i, len;

    t = tree;
```



```
    len = strlen( morse );
    for( i = 0; i < len; i++ ) {
        if( morse[i] == '.' ) {
            if( t->dot == NULL )
                t->dot = morsetree('*');
            t = t->dot;
        }
        else {
            if( t->dash == NULL )
                t->dash = morsetree('*');
            t = t->dash;
        }
    }
    t->letter = letter;     read    return(t->letter)
}

char read_morse(Morsetree *tree, char *morse)
{
    /*
     * You write this code to traverse the tree based on
     * the Morse code sequence of dashes and dots. Return
     * the letter found at the final node of the tree.
     */
}

int ReadLine(char *buff, int size, FILE *fp)
{
  buff[0] = '\0';
  buff[size - 1] = '\0';              /* mark end of buffer */
  char *tmp;

  if (fgets(buff, size, fp) == NULL) {
      *buff = '\0';                    /* EOF */
      return false;
  }
  else {
      /* remove newline */
      if ((tmp = strrchr(buff, '\n')) != NULL) {
        *tmp = '\0';
      }
  }
  return true;
}

char *multi_tok(char *input, char *delimiter) {
    static char *string;
    if (input != NULL)
        string = input;

    if (string == NULL)
        return string;

    char *end = strstr(string, delimiter);
    if (end == NULL) {
        char *temp = string;
        string = NULL;
        return temp;
    }

    char *temp = string;

    *end = '\0';
    string = end + strlen(delimiter);
    return temp;
}
```