**Developers documentation**

**“Morse code” console application in C**

**Name:** Munkh-Orgil

**Requirements:**

1. File handling should be used
2. Program should consist of several modules
3. Appropriate data structure, i.e. binary tree should be used to code and decode
4. Program should be controlled by command line switch.
5. Program must be satisfied to the provided specification

**Modules:** this program includes 3 modules, which are 2 source files (main.c, func.c) and 1 header file (func.h). This provides for the program to be more readable. In the main.c file there is a main function (control menu, binary tree table), in the func.c file there are all the functions that are used by the program and func.h file contains the definitions of all those functions and structures.

**Data structure:**

This structure is a binary tree. It saves one letter and have left and right pointers.

*• typedef struct btree {*

*char letter;*

*struct btree \*r,\*l;*

*} btree;*

This structure is saving a letter, corresponding morse code and counter of letter, which will be used in giving statistics.

*• typedef struct letter\_morse {*

*char letter;*

*char morse[7]; // there at most 7 characters in morse alphabet*

*int l; } Trsnlt;*

**Functions:** In total 16 functions were used in this program.

***Void Printmenu()*** function prints the main menu, it is void function, since has no any input parameters.

***char \*read\_long\_line()*** *{ //this function reads the text until the new line*

***void morse (char \*text, Trnslt \*s)*** */\* this function finds from the translate table and prints the appropriate morse code to the latin character for each letter of the text\*/*

Morse is void type function which return nothing, but printing coded text into text file and on the screen. The function takes pointer to strings and pointer of typedef Trnslt. It opens “encode.txt” file and appends some extra data into it. If pointer parameter is not NULL pointer we takes following actions. By for loop, function goes through all characters of string and comparing it with ‘letter’ variable of structure Trnslt, by going through all array of structure. The reason for both situation I chose to use for loop is because it’s clear how many steps need to be completed. If character from string pointer and ‘letter’ variable of structure matches the function prints corresponding morse code, which is in the structure, and it also prints it into opened text file. After use we close text file to not lost some data from there.

***void code (Trnslt \*s)***

Code is void type function which combines read\_long\_line and morse functions. At the end, it frees used dynamic array created by read\_long\_lin function to prevent overuse of computer’s storage.

***btree \*insert (btree \*root, char c, char \*s)***

Insert is btree type function which returns head pointer of binary tree. It takes as parameter pointer of binary tree(root), character(c) and pointer to string(s). Since it’s recursion function there must be base statement. If root equal to NULL pointer it creates new node to binary tree and temporarily place there space and equalize left and right nodes to NULL pointer. Until terminator of string s, function moves through tree, if dot to left, if dash to right, by recalling itself. When it reaches to terminator, insert function place character c there.

// this function will insert all my alphabet which is in array of Trnslt.

***btree \*Allinsert (btree \*root,Trnslt \*s)***

Allinsert is btree type function which takes pointer of binary tree and pointer of structure as parameters. It goes through tree and ,by instert function, place all variable of array of structure and returns modified pointer of binary tree.

***void search (btree \*root,char \*s)***

Search is void type function which means it will return anything. It takes pointer of binary tree and string as parameters. As usually the function initializes f file. It opens “decode.txtx” text file and append some data into it. The function will go through the string, which is Morse code, and search for corresponding letter by movie left (point) and right (dash) nodes of tree. If it reaches terminator, this node is what we searching for and function will print a letter that’s on that node into the text file and on the screen. Also prints space and \n where it’s needed. At the end function close text file f to prevent memory leak.

***void decode (Trnslt \*s)***

This function combines 3 functions need for decoding: read\_long\_line, Allinsert and search.

***void statisticDecd (Trnslt \*s)***

StatisticDecd is void type function. This function will read “decode.txt” text file until end of file and search through array of structure for matching letter. If it founds matching increases structure’s l value which counts how many times this letter is used.

***void statisticEncd()***

This is void function, which prints statistics of usage of dashes and dots. Opens “encode.txt” text file and reads from there all data until end of file. At the end, closes file to prevent memory leak and deletes “encode.txt” file to start next from new file.

***void playGame (Trnslt \*s)***

This function gives an opportunity for user to check his knowledge on morse alphabet by playing a short “game”. Program asks the user to enter any letter he wants, then asks to translate it into morse and enter the result too. Program takes latin later, compares is to translate table and then saves the correct encoded answer in the string. Then program compares correct answer with the encoded version of the user and if it is the same, the appropriate message is printed. Game can be continued again and again, until user press 0 to exit the program. The playGame returns nothing as well, that is why it is a void type function.

***char \*read\_from\_file (char \*filename)***

The parameter of the function is a filename of the input file. Function reads the file and converts it into string. At the end function returns the string.

***void morse\_file (char \*text,Trnslt \*s, char\* saving\_file)***

This function takes a translate pointer and 2 strings (the one that we want to encode and a string which contains the name of file we want to save it in) as parameters. The function is similar to morse() function. As a result we get a file with encoded text.

***void encode\_file (Trnslt \*s)***

It is a combination of 3 functions: the one which *reads a string* until new line(in our case it’s a filename), 2nd *reads from file* and copies it into string and *morse\_file* function which takes the string, the output filename and gives the encoded file

***void search\_file (btree \*root,char \*s, char \*saving\_file)***

This function searches letters 1 by 1 from binary tree of morse code and prints it into file that user want to save it in. Similar to search() function described above.

***void decode\_file (Trnslt \*s)***

This function takes a translate pointer and combines several functions (read line, read from file, all insert and search file). As a result we get a file with decoded text named by user. Similar to decode() function.

***int main ()***

Main function is where all the functions are called and it’s the most important part of program. Here initializes all the corresponding morse code for each letter in the Trnslt structure. Switch is used to control the program. In the case 1, program calls code function. In the case 2, it calls decode function. In the case 0, it calls statisticDecd and statisticEncd functions. In the case 4, it calls encode\_file function. In the case 5, it calls decode\_file function. In the default, when user enters wrong input it prints warning. To do it several times switch is the while loop. Program stops working when user enters “0”.