Budapest University of Technology and Economics

Programming 2 (Course code: BMEVIIIAA03)

Academic year: 2020/21/2

Final Project: Morse Converter (Windows)

Name: Tushig Bat-Erdene

Neptun ID: QBI3JH

8th of May 2021

Contents:

Introduction:

Morse code is mainly used in telecommunication to encode text characters by standardized sequences of different signal durations (as known as dots and dashes). In this paper, I will be explaining the main functionality of this program and describing my solution to understand how exactly it works.

Main functionalities of the program:

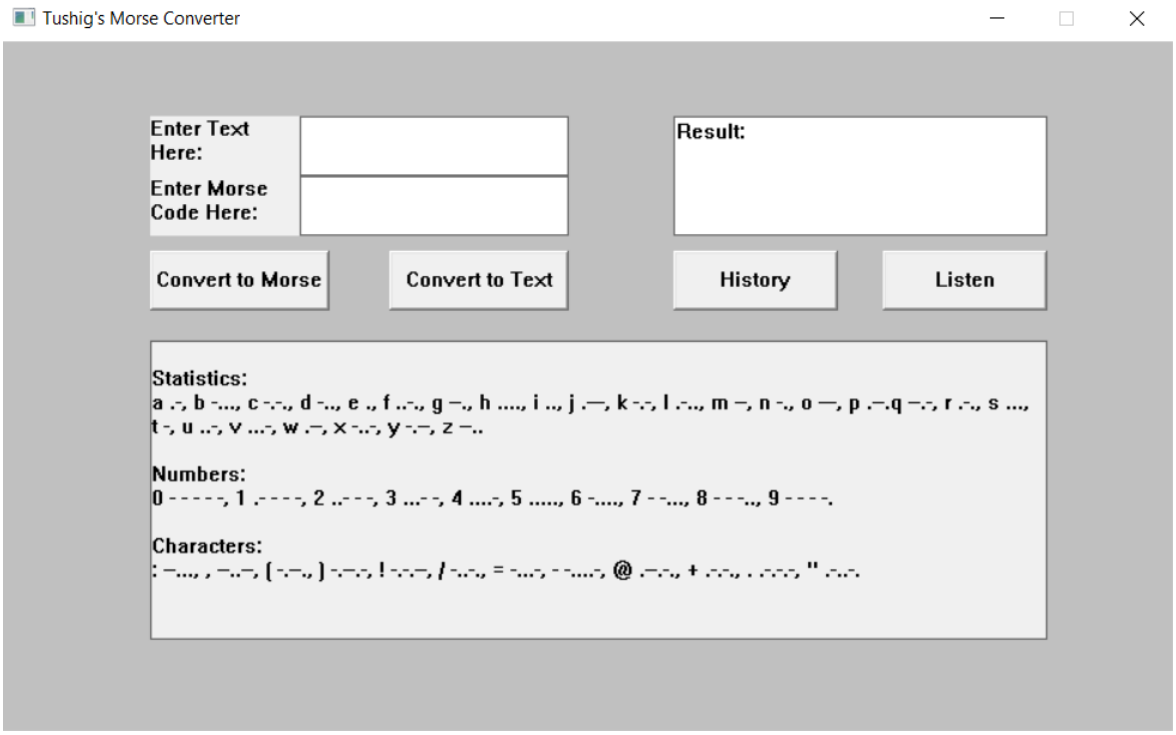

Figure 1. User Interface of the program

According to the Figure 1, It displays the main user interface of the program. Following operations will be the functionalities of it:

- Enter Text Here: This text box gets the string text from the user. It only read lowercase characters.
- Enter Morse Code Here: This text box gets the morse code from the user. It only read morse code consists of dots and dashes.
- Result: This text box displays the output of conversions based on the input from the previous two text boxes.
- Convert To Morse: This button has the functionality which is making it possible to convert user input, which has gotten from “Enter Text Here:”, and displays the conversion in the “Result:” box. If the user input is blank or empty, it throws an exception.
- Convert To Text: This button has the functionality which is making it possible to convert user input, which has gotten from “Enter Morse Code Here:”, and displays the conversion in the “Result:” box. If the user input is blank or empty, it throws an exception.

- History: This button displays the “Alert!” message box which points the user to the location of the “history.txt” file.
- Statistics box is the static box which contains the translation of characters in Morse code.

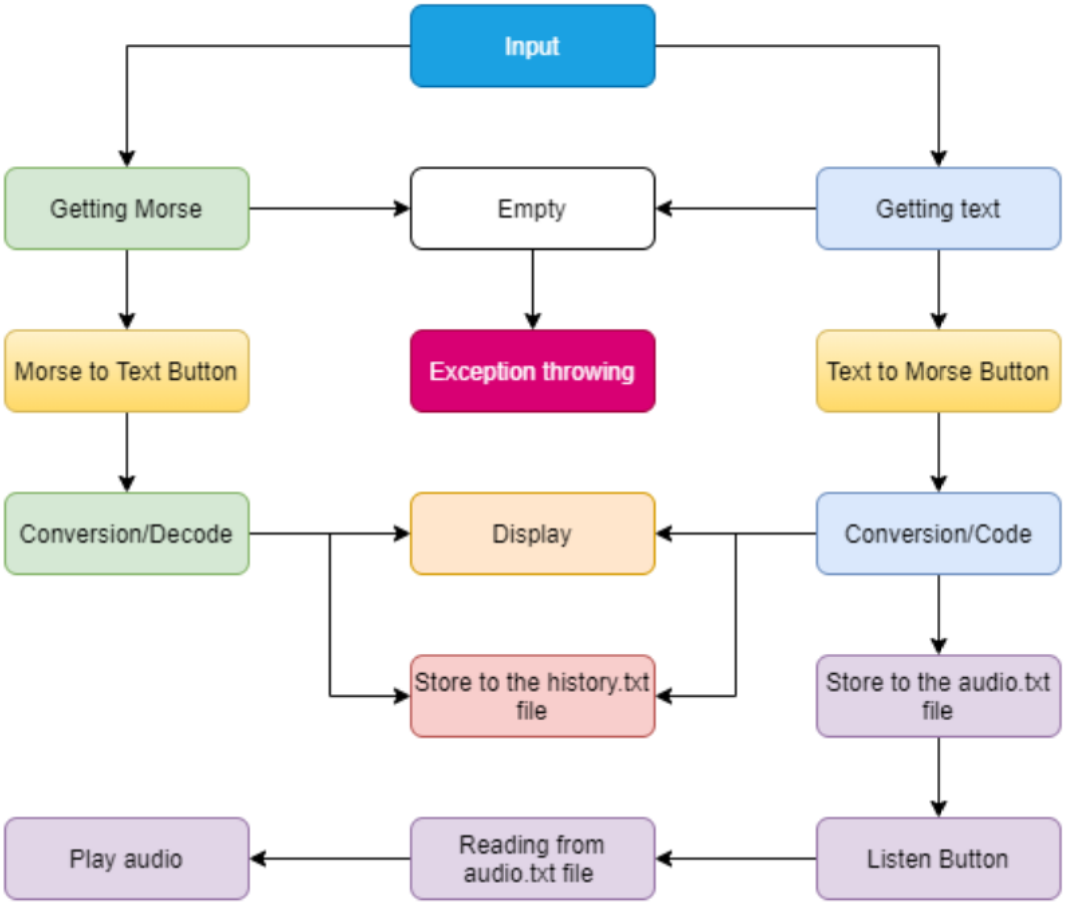

Figure2. Simple chart of basic operation

Requirements to build the program:

- Microsoft Visual Studio or any other integrated development “IDE” which can compile C++ which must containing “Windows sub-system” because of the program cannot compile itself in the console application.
- Those libraries are included:
  - #include<windows.h> → which is used for initializing window and audio
  - #include<iostream> → which is used for input and output operations
  - #include<fstream> → which is used for file handling or management
  - #include<tchar.h> → which is used for some characters because this program is not built in UNICODE enabled.

    - #include<string> → which is used for operating with strings also it is important to manipulate character arrays.
- Before compiling the program, please set up following settings:
    - Go to the “Solution Explorer” -> Click mouse 2 as shown in the picture -> “Properties”
    - Go to “Configuration Properties” -> “Advanced” -> “Character Set” -> Set to “Not Set”
    - Go to the “Linker” -> “System” -> “SubSystem” -> Set to “Windows”
    - Press “Apply” -> “OK” Doing this setting is making it possible to compile without error because MS Visual Studio cannot compile windows application in console settings.

Data structure of the program:

I used linked list and binary tree for my converter because it is the simplest way to represent morse characters by connection of its nodes. It has at most 2 parent nodes “left, right” and it is making it possible to initialize morse library easier.

As you can see from the following figure, I made left subtree of a nodes are the morse codes which are starting with dashes and right subtree of a nodes are starting with dots. For instance, finding “g” is Start → left → left → right.

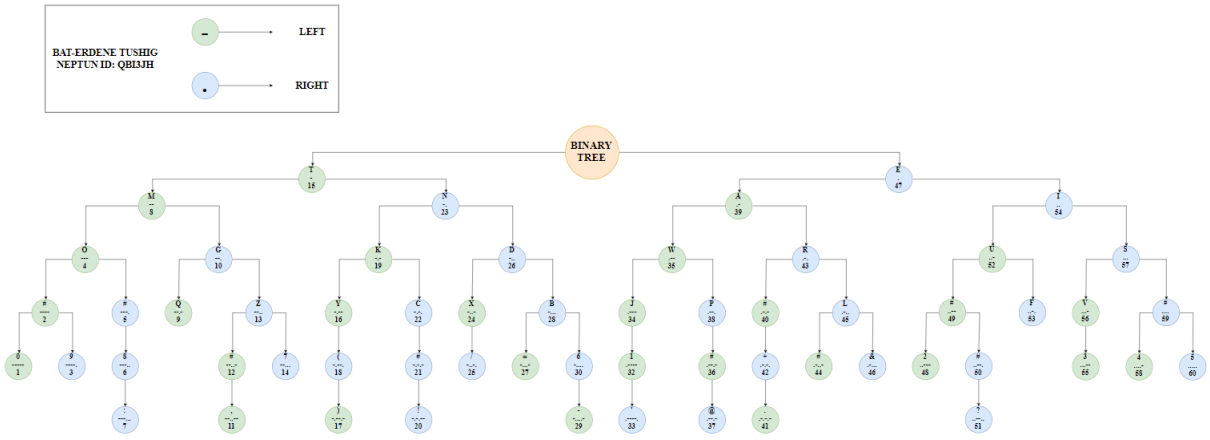

Figure 3. Binary Tree Diagram

Techniques used for the program are following:

- Object oriented approach → It is realized with:
    - Header files:
        - Morse.h
        - Audio.h
        - History.h
    - Implementation files:

    - Morse.cpp
    - Audio.cpp
    - History.cpp
- Dynamic Memory Management → It is realized with the essential functions in the Morse.h class and its implementation.
- File Management → It is realized with the History.h class and its implementation which storing the conversions in the history.txt file in the folder where this project or program is located. Also, Audio.h contains one function which is using file management to create audio.txt file and read from it.
- Exception handling → It is realized in the main.cpp file for getting the user input. For instance, if the text box which is getting text or morse input is blank it throws an exception.

Functions:

The following functions are in the main conversion class (Morse.h, Morse.cpp):

1. **Typedef struct BinaryTree**
   I implemented this function by using linked list data structure which is making it possible to search characters from library.
2. **Typedef struct MorseLibrary**
   I also implemented MorseLibrary function by using linked. It contains characters and string has MorseCode inside it. It is also making it possible to creating array which contains all characters and its translation in morse code.
3. **BinaryTree* Morse::Node(char item, int key)**
   This pointer function is for putting characters in the BinaryTree by creating nodes for each character.
4. **BinaryTree* Morse::AddNode(BinaryTree* start, char value, int key)**
   This function can insert characters by depending on the key value which goes left or right.
5. **BinaryTree* Morse:Morse()**
   I created a well-ordered binary tree diagram “Figure 3.” which helped me a lot by defining characters and morse codes clearly. This function is generating a pointer points start (start is a root of binary search tree in my program) with the help of Node function. It is inserting all available characters to tree using AddNode function. It works like If you go left from the root (I used start to demonstrate root) you will find dash (- , which is character T), if you go to the right you will find dot (. , which is character E) and it will return to the start again. It decides whether go left or right by depending on the key value which you can see it from the diagram from the first page of this document.
6. **char Morse:GetText(const char* str)**
   This function has a role to get string from a user to convert morse code into text (More like character finder). It measures the size of string also creating the pointer from start root to get characters. As you can see the for loop to check whether it is left or right. If user inserts the morse code which is invalid or do not belong in the library, the program

will say “Invalid Code!” (It is only for the console application I will try it work for the windows application).

7. **const string Morse:MorseToText(const char* str1)**
   This is the one of the essential functions for converting morse into the text. It has a big role to convert morse into the text. This function is getting string and calling its translation from the constructor to convert. For now, it is only printing function, and it is void which is returning nothing. I will make it string function and return string because I need string output to display it in morse window.
8. **const string Morse:TextToMorse(const char* txt)**
   This is the essential function for converting text into the Morse code. It is getting string and length of text to find translation. It is also returning the string as an output because of making it possible to display in the window.
9. **void Morse:statistics()**
   This function is for just checking if it is printing the statistics the translation of English and Morse codes.
10. **void Morse:deleteNode(BinaryTree *root)**
    This function is deleting nodes from the binary tree until root will NULL.

The following functions are in the history class (History.h, History.cpp):

1. **void History::createHistory()**
   This function is for initializing history.txt file.
2. **void History::AddToHistory()**
   This function is for appending string to the history.txt file.
3. **void History::lineBreak()**
   This function is for appending lines into the history.txt file to display the results clean.
4. **void History::closeFile()**
   This function is for closing the history.txt file.

The following functions are in the audio class (Audio.h, Audio.cpp):

1. **void Audio::dotAudio()**
   This function is initializing a sound for dot with the help of Beep() function from windows.h library.
2. **void Audio::dashAudio()**
3. This function is initializing a sound for dash with the help of Beep() function from windows.h library.
4. **void Audio::pauseAudio()**
5. This function is pausing a sound with the help of Sleep() function from windows.h library.
6. **void Audio::MorseBeeper()**
   It is the main function that is making it possible to create sound from the string by finding dots and dashes.
7. **void Audio::listen()**
   This function is initializing listen text file to store morse code because of windows application cannot get the previously converted morse code by passing it within the cases.

Main.cpp file → It is the main file contains windows application codes:

1. **#define _CRT_SECURE_NO_WARNINGS**
   It is removing preprocessor secure warnings because this is not written in UNICODE.

2. **Button definitions:**
   #define LISTEN_BUTTON
   #define HISTORY_BUTTON
   #define CONVERT_BUTTON
   #define NEWCONVER_BUTTON
3. **Void AddButtons(HWND hWnd)**
   In this function I initialized every text box windows and buttons.
4. **Case WM_CREATE:**
   This case is for windows creation and I called the AddButtons(); function to made user interface “Figure 1.”.
5. **Case WM_COMMAND:**
   In this case I set up operations of functionalities for all the buttons.
6. **Case CONVERT_BUTTON:**
   It is for getting text in the text box and coding it to the morse code. Also, it is storing them to the history.txt and storing only converted morse in the audio.txt file. Exception throwing also initialized in this case.
7. **Case NEWCONVERT_BUTTON:**
   It is for getting morse code in the text box and decoding it to the text and storing them in the history.txt. Exception throwing also initialized in this case.
8. **Case HISTORY_BUTTON:**
   Showing the message box which is displaying the message about its location.
9. **Case LISTEN_BUTTON:**
   It is reading the audio.txt file which generated in the case CONVERT_BUTTON and playing the information using Beeper().

Testing:

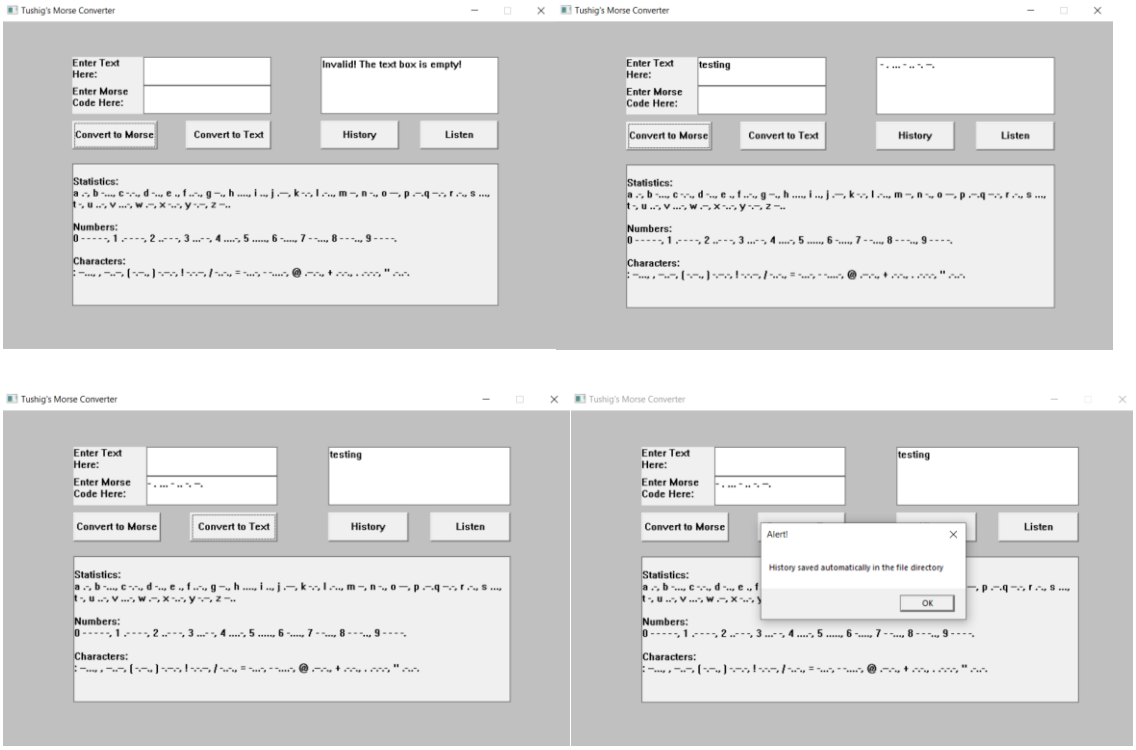

Figure 4. Testing

Testing “Convert To Morse” button, successfully converted.

Testing “Convert To Text” button, successfully converted.

Testing exception handling of “Convert To Morse, Convert To Text” buttons, successfully threw “Invalid! The text Box is empty!”.

Testing “History” button, successfully displaying the “Alert!” mini-window message.

Testing “Listen” button, successfully playing the morse code.

Testing file management, both history and audio files have exact results.

References:

[1] http://eik.bme.hu/~csurgai/Mswin_en/Mswin_en.htm

[2] https://www.cplusplus.com/reference/string/string/c_str/

[3] https://www.dropbox.com/s/k7fb15c3f1o4yf8/forgers-win32-tutorial.pdf?dl=0

[4] https://docs.microsoft.com/en-us/windows/win32/learnwin32/winmain--the-application-entrypoint

[5] http://www.winprog.org/tutorial/controls.html%20

[6] https://www.youtube.com/watch?v=FXrulJXMiU8

[7] https://www.youtube.com/c/ThePentamollisProject/videos

[8] International Morse Code | Morse Code World