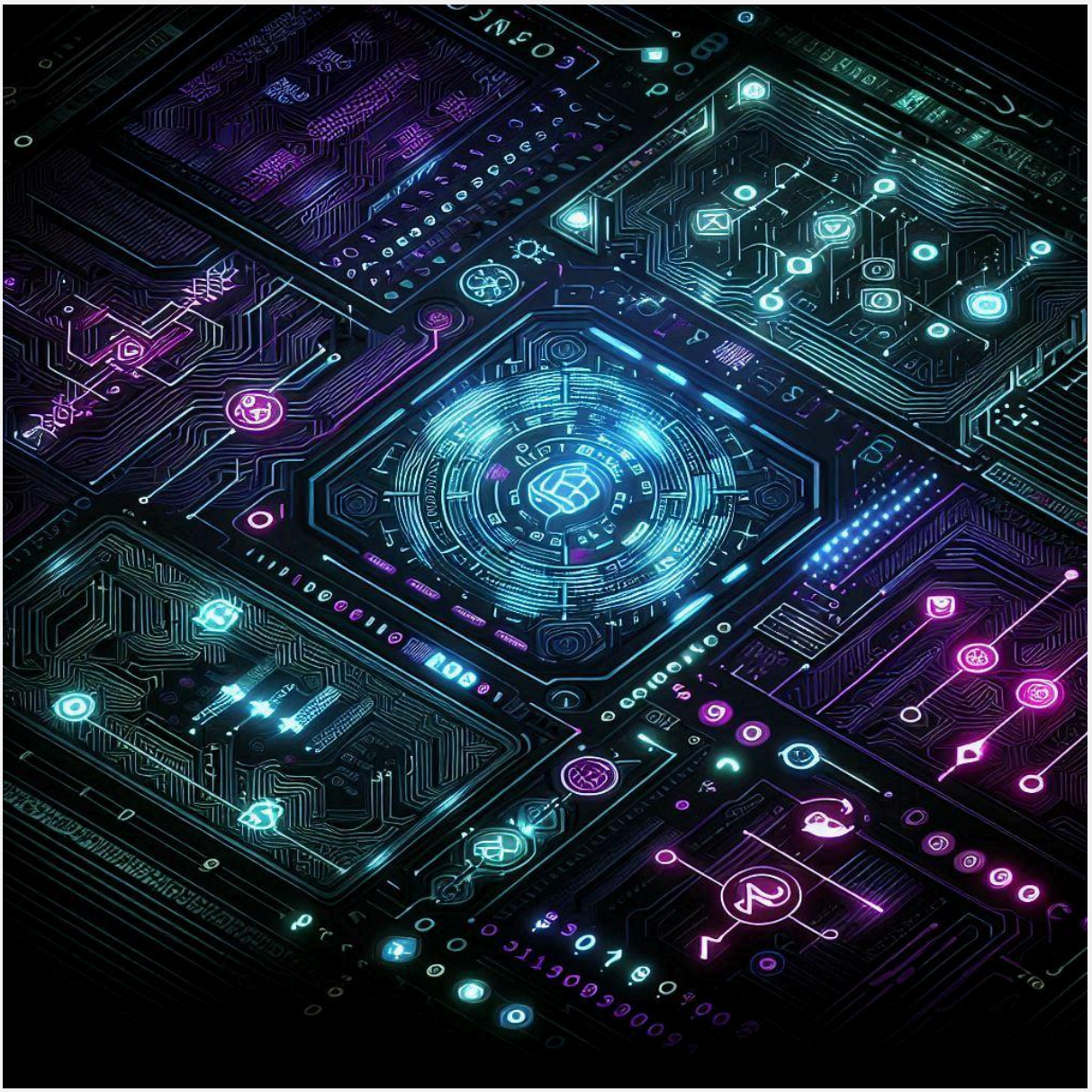
BBM203: Software Practicum - I, Fall 2024
Practice Assignment, Topic: File IO



Prof. Dr. Emirhan Yalçın

.--. .-. --- ..-. .-.-.- / -.. .-. .-.-.- / . -- .. .-. .... .- -. / -.-- .- .-.. -.-.. .. -.

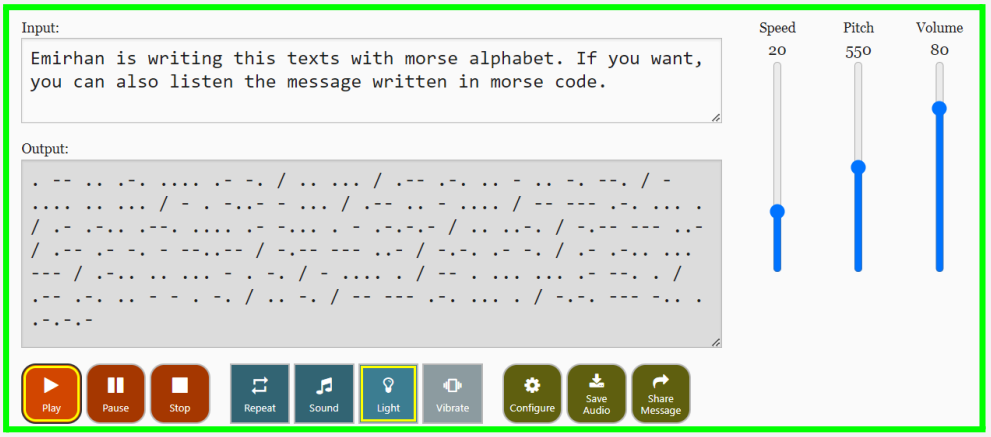
1. Introduction

In this project, we focus on implementing a **Morse code converter** using C++. The program will allow users to convert English text to Morse code and decode Morse back to text, demonstrating the practical use of the Morse alphabet in programming. You will simulate an encoding-decoding procedure between students about their instructors.

2. About the Morse Alphabet

The Morse alphabet is a method of encoding letters, numbers, and symbols into sequences of **dots and dashes**. Developed in the early 19th century for long-distance communication, it was widely used in telegraph systems. Each letter or number is represented by a **unique combination of short and long signals**, making it a simple yet effective way to transmit information.

Below you can see an example of the converted morse code:

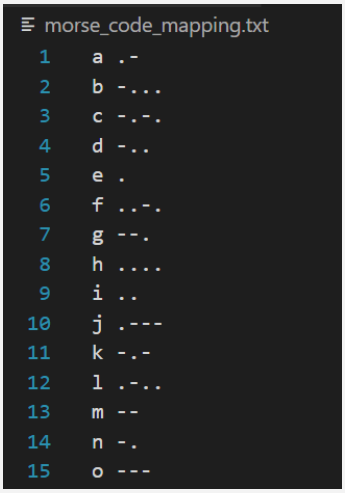


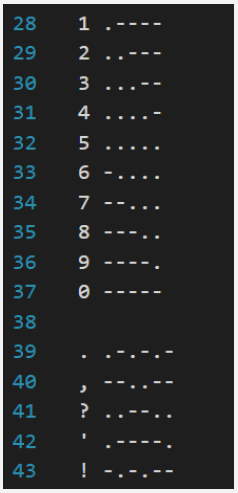
A morse text consists of three different characters. **‘.’ ‘-’ and ‘/’**. Note that the slash character ‘/’ is used for word spaces, not being the part of the alphabet directly. You can visit the link if you want: https://morsecode.world/international/translator.html

3. Morse Translation Table

The mapping between English letters and the morse alphabet is given to you in the **morse_code_mapping.txt** file. You are expected to read this file using file streams of C++ and convert it to a dictionary-like structure for later use. This is the first thing to do in your practice assignment.

Morse text is case insensitive. In your mapping file, they are given
In the lower case, all English letters have the same morse code for all cases.





As you may see some screenshots from your ***`morse_code_mapping.txt`*** *, each character of the English alphabet, digits and punctuations are followed by their correspondence in the morse script.*

We can consider the output of the morse alphabet conversion as a signal of a single line. All dots and dashed are printed in a single line consecutively, with the space character **"/"** between them.

4. The Story, Translations and Some Rules to Consider

A group of BBM and AIN decided to use morse coding for gossiping about their friends and their instructors, since it is much safer. The students transfer their secret messages to each other using txt files. Files contain the morse script that are **encoded** to English. And of course, someone needs to **decode** the morse script they've received, since all the dots and dashed normally doesn't mean anything for humans.

You should develop a program that will take some **.txt** file as the first **command line argument** and the **target "code"** for translation, like a flag for direction of the translation. Your program needs to both encode the given English text and also decode the given morse script. As it says, you need to implement your program in a way that accepts command line arguments. For example:

- **./program <file_name.txt> <morse|english>** is the template for command line arguments of running your code.
- **./program english_input.txt morse** will convert the english text in the input file to morse code and write it into another new-created txt file.
- **./program morse_input.txt english** will convert the morse code in the input file to English and write it into another new-created txt file.

Normally, your program should add the converted script-name to the end of the input file name when creating a new file. For example, if you convert **shakespeare.txt** from English to morse, then your output file should be named as **shakespeare_morse.txt**.

However, the students also gossip about their instructors and TAs, also SAs, which makes their communication risky. They are afraid of getting caught by their instructors. For this reason, they developed a genius technique. If the name of any instructor appeared in the file they are encoding or decoding, they would save the output file with a different format to avoid drawing their teachers' attention. They decided to name the output file as follows to make it look like an ordinary system file and not attract their teachers' attention:

`system_data_dll_{reversed_input_file_name}_morse.txt` **or**
`system_data_dll_{reversed_input_file_name}_english.txt`

If a file contains the name of any instructor, assistant, or student assistant assigned to this course, either written in Morse code or in English, your program should automatically save the file in this output format. As you may have noticed, output file name is not a parameter given to your program. For simplicity and to avoid issues with Unicode, you can assume that all instructor names are written using the English alphabet and do not contain Turkish characters.

Consider the names : `['selma', 'asli', 'meryem', 'engin', 'hacer', 'adnan', 'emirhan', 'anar']`

The file **morse_code_mapping.txt** is static. No need to take an argument for this mapping file’s name. You can keep this as constant. In your mapping file, lower case English letters are used. In the decoding part, conversion from morse to English, you can print everything in lowercase. And also again create a single line output, putting every decoded word on a single line.

5. Helper Functions Given

Since I don't want to over-complex the implementation for you, here I am sharing the implementation of some string / character operation functions for you. You may or may not use them in your implementation. You can find them in HelperFunctions.cpp. You can directly use them or adapt them into your implementation.

Helper methods

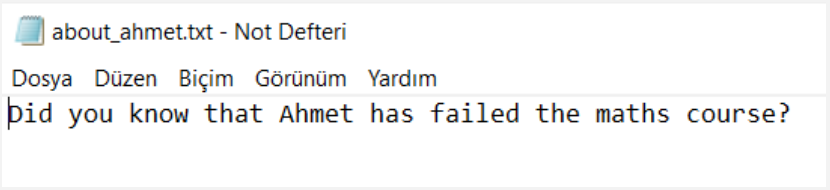
```cpp
#include <string>
#include <algorithm>

// Convert a single character to lowercase
char toLowerCase(char ch) {
    return std::tolower(static_cast<unsigned char>(ch));
}

// Convert a whole string to lowercase
std::string toLowerCase(const std::string& str) {
    std::string result = str;
    std::transform( result.begin(),
                    result.end(),
                    result.begin(),
                    [](char ch) { return toLowerCase(ch); });
    return result;
}

// Convert a single character to uppercase
char toUpperCase(char ch) {
    return std::toupper(static_cast<unsigned char>(ch));
}

// Convert a whole string to uppercase
std::string toUpperCase(const std::string& str) {
    std::string result = str;
    std::transform( result.begin(),
                    result.end(),
                    result.begin(),
                    [](char ch) { return toUpperCase(ch); });
    return result;
}

// Reverse a given string
std::string reverseString(const std::string& str) {
    std::string result = str;
    std::reverse(result.begin(), result.end());
    return result;
}
```

6. Sample File IO

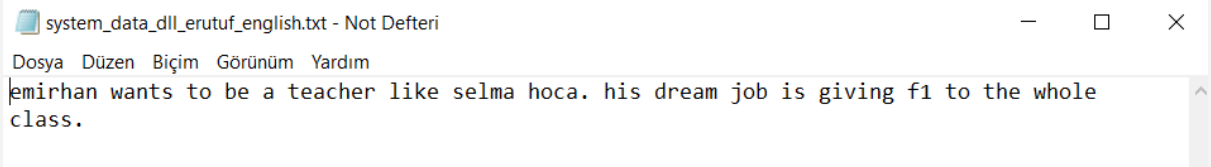
For the English input files, the text can contain both lower and upper case characters, including numbers and punctuation marks. Especially consider the existence of the lower and upper case letters in your implementation.

Morse code doesn't have a distinction between lower and upper case letters. When you are decoding, from morse to English, you can write every character in lower case. Note that basically all output files that your programs need to be in a single line. And also you make a translation word by word. You may discard new line characters and empty lines etc. This may give you an insight about the file reading method you can use.

Below you can see a translation done from English to Morse. The second file actually consists of a single line. Due to visualization techniques of notepad application, it may seem like multiple-line file.



And below there is a morse to English translation. Note that the file contains an instructor name and different naming for the output file is needed.



5 English and 5 morse input files will be provided to you for this task.

7. Run Format and Make File:

As it has mentioned earlier, your program accepts two command line arguments. first is the input txt file name and the second one is <morse | english>, the translation direction. Also you will be provided a makefile, which makes your job probably easier.

- **“make”** command will compile your files.
- **“make run”** will run your code with 10 sample input and output files.
- **“make clear”** will delete the compiled program and generated output files.

Check piazza post to see the makefile. You can configure the makefile if you want to try your program with different input output files.

8. Final Notes:

- This is an ungraded project. You are free to implement, but strongly advised.
- For this ungraded project, you are free to share your codes and ideas with your friends.
- You may get help from AI tools. But be sure you use AI tools in a way that they help you **learn**, not directly implementing and solving your problem.
- You can ask your questions through piazza. If you think there is some error etc. you can inform me.
- Happy Coding!