CCNU-UOW

CSCI851 Advanced Programming
Autumn 2020

Prof. Zhifeng Wang

Assignment 3 (Worth 10%)

Due 11:55pm December 13^{th} 2020.

This assignment involves implementing container and class functionality to support translation between symbols sets using a provided rule. The symbol sets that need to be supported for the assignment are associated with Morse code and Braille.

General notes

These are some general rules about what you should and shouldn't do.

1. Your assignment should be sensibly organised with the same kind of expectations in this area as the previous assignments. No memory leaks etc. too.
2. Your code must compile on Banshee with the compilation instructions you provide in `Readme.txt`. If it doesn't you will likely receive zero for the coding part of the assignment.
3. **Other than the initial command line input, the program should run without user input.**
4. **If there is a problem with the data at read time, such as an invalid file, missing file or incorrect symbols, report the problem and abort.**
5. **If there is a problem with the translation of data report the problem and display the translation up to that point, and then abort.**
6. You have to use classes for this assignment, but you should not use inheritance.
7. You should use templating for this assignment.
8. Failure to comply with any of the two points above will result in significant penalties.

Translation

Once your program is compiled into the executable `TS`, it must run as follows:

```
./TS From To In.txt Out.txt
```

where `From` and `To` indicate the source and target alphabets, either M, B, or L, and the two values should be different or you should report an error. `In.txt` will contain an input data sequence, in the source alphabet specified in `From`, while your `Out.txt` program should generate the output data sequence in the target alphabet specified in `To`.

Alphabet Classes

You need to write classes to support the following alphabets, one class for each. These classes should not be related by inheritance.

M : Morse. Morse uses sequences of dots (.) and dashes (-) to represent letters and numbers. We are only interested in letters here, and a space symbol. The translation rule for Latin to Morse is given in the file `Morse.txt`. In that file we use underscore (_) to represent a space in Latin text and use four dashes to represent that, which is not standard Morse. Note that you need to use a space to seperate the encoding of different Latin symbols into Morse.

B : Braille. Braille uses a 2 by 3 grid of flat or raised positions. This can be represented in a binary string. Again we are only interested in letters and a space symbol so can use a 6 bit string. The translation rule for Latin to Braille is given in the file `Braille.txt`. In that file we use underscore (_) to represent a space in Latin text and use 000000 to represent that.

L : Latin. The allowed symbols are the standard lower case letters used in English and a space ( ).

Those translation files can be found in `/share/cs-pub/251/Assignments/Three`. That directory also includes some sample data files. `Latin1.txt`, `Morse1.txt`, and `Braille1.txt`, all represent the same message. Note the different meaning of spaces in Morse to Latin and Braille. Note also that to convert between Braille and Morse your program needs to go via Latin. There are fixed names for the translation files, but the content may not always be the same so don't hard code the content into your program.

Upper case letters for Latin should be considered to be invalid data.

An instance of one of the alphabet classes contains a single valid symbol for that class.

A container : A class template.

You need to write a container that will hold a collection of objects of an appropriate alphabet. One instance of this container will hold the elements read in from `In.txt`, another will be in a different alphabet and will contain the sequence of symbols generated by translation and ready for output to `Out.txt`. If you are translating between Morse and Braille you will need an additional container instance to store the intermediate Latin sequence.

The process

Once the user has run ...

```
./TS From To In.txt Out.txt
```

you need to prepare for dealing with the appropriate alphabets. The translation rules should be tied to the Morse and Braille classes and you should validate and report on their loading. Validation involves making sure only appropriate Morse/Braille symbols are used and that every Latin letter is mapped uniquely.

Once the translation rules are loaded you can start reading the input data from `In.txt`. You should be checking each symbol is consistent with the container being read into as you go.

Once you have completed reading from `In.txt` you should report on the container content. This is the overall length of the container and the symbol distribution within it.

You then carry out translation. Once the translation is complete you output the resulting container content to standard out, and again report on the overall length of the container and the symbol distribution within it. If this container is the final one, you output the content of the container to `Out.txt`. If the container is an intermediate Latin one, you carry out another translation to the final alphabet, report on the length and symbol distribution again, and finally output the resulting sequence to `Out.txt`.

If there is a problem with the translation of data report the problem and display the translation up to that point, and then abort. This will likely happen if you have an incorrect string of valid symbols in the file you are reading from.

Notes on submission

Submission is via Moodle.

Your code must compile on Banshee with the instructions you provide. If it doesn't you will likely be given zero for this assignment.

Please submit your source, so .cpp and .h files, and Readme.txt file, and makefile if you have one, directly to Moodle. There shouldn't be other files or directories and they shouldn't be in a zip file. Submission is via Moodle.

1. Late submissions will be marked with a 25% deduction for each day, including days over the weekend.
2. Submissions more than three days late will not be marked, unless an extension has been granted.
3. If you need an extension apply through SOLS, if possible **before** the assignment deadline.
4. Plagiarism is treated seriously. Students involved will likely receive zero.