**FreeNeb Embedded ASR Engine Test Specification**

**Introduction**

This is a test for FreeNeb embedded ASR engine, which can convert the input audio to text. The engine contains a header, an so file, an example, a document, relative source code, and some files which will not be delivered. User is allowed to imitate the example to write some code to invoke the four functions in header, and implement functionality. But the user has to follow some restrictions. The test aims to check the engine, estimate function and performance, and find severe bugs.

**Test range**

inclusion

1. Check if the document is complete and clear
2. Check if the example can be implemented properly
3. Check if the engine can get output text
4. Check if the provided AM and LM will cause crash
5. Check with normal case and marginal case under restrictions.
6. Check if the engine will crash
7. Check if there is memory leak
8. Check if the output is random

exclusion

1. Do not check the accuracy of the result
2. Do not check if the engine will get a result when input is out of restrictions
3. Do not check the circumstance when AM or LM is not the designated ones.
4. Do not check files which will not be delivered( e.g. wer computing script)

**Test method**

1. Read the document, and implement each instruction.
2. Run the example.
3. Modify the example, use normal input and marginal input, watch the output.
4. Repeat the same code several times and compare them
5. Keep the engine running continuously and inspect the memory usage.
6. Use testing tool valgrind to check memory leak

**Test case**

1. Read document, check if the restrictions are clear.
2. Install the engine as instructed.
3. Check if the example can run properly
4. Check when using nnet3 AM, general LM, and reading audio segmentally
5. Check when using nnet1 AM, command LM, and reading audio segmentally
6. Check when using nnet3 AM, general LM, and reading audio in whole
7. Check when using nnet1 AM, command LM, and reading audio in whole
8. Check when audio segment length is 1
9. Check when audio segment length is large
10. Check when audio segment length gets the maximum
11. Check when audio segment length is dynamic
12. Check when audio segment length is power of 2
13. Check when audio segment length is odd
14. Check when audio segment length is even
15. Check when input extra long audio, and read it segmentally.
16. Check when input extra long audio, and read it in whole
17. Check when input extra long audio, and read it with small segment length
18. Check when input short audio, and read it segmentally
19. Check when input short audio, and read it with big segment length
20. Check when input short audio, and read it with small segment length
21. Check when input loud audio
22. Check when input whisper audio
23. Check when input silence audio
24. Check when the engine runs continuously
25. Check when input extra long audio, and run continuously
26. Check when read audio in whole, and run continuously
27. Check when audio segment length is small, and run continuously
28. Check when audio segment length is dynamic, and run continuously
29. Check the engine with valgrind
30. Check when input extra long audio with valgrind
31. Check when read audio in whole with valgrind
32. Check when audio segment length is small with valgrind
33. Check when audio segment length is dynamic with valgrind

**Appendix**

restrictions:

There are four functions for users to invoke

1. for function decoder\_put\_data(struct decoder\_obj \*ptr, const char \*data, int len, int chunk\_no):

size of data must be equal to len,

len must be greater than 0,

chunk\_no must count as the sequence [1, 2, …, n-1, -n],

data must come from a wav file, which is 16kHz in sample rate, 16 bit in depth, and is mono channel.

1. for function decoder\_get\_result(struct decoder\_obj \*ptr, char \*result, int len):

length of result must be equal to len,

len must be greater than 0