In [1]:
```
import pickle
import math
from nltk.tokenize import word_tokenize
from nltk.util import ngrams
```

Each bigram's probability with Laplace smoothing is: (b + 1) / (u + v) where b is the bigram count, u is the unigram count of the first word in the bigram, and v is the total vocabulary size (add the lengths of the 3 unigram dictionaries).

In [2]:
```
def unpickler(language):
    Unigram = pickle.load(open(f'{language}_Uni.pickle', 'rb')) # read binary
    Bigram = pickle.load(open(f'{language}_Bi.pickle', 'rb'))
    return Unigram, Bigram
```

In [3]:
```
def calc_prob(each_line, unigram_dict, bigram_dict, V):
    test_unigrams = word_tokenize(each_line)
    test_bigrams = list(ngrams(test_unigrams, 2))

    p_laplace = 1 # laplace smoothing

    for each_bigram in test_bigrams:
        n = bigram_dict[each_bigram] if each_bigram in bigram_dict else 0
        d = unigram_dict[each_bigram[0]] if each_bigram[0] in unigram_dict else 0
        p_laplace = p_laplace * ((n + 1) / (d + V))

    return p_laplace
```

In [9]:
```
def main():
    # Read in your pickled dictionaries
    E_Uni_dict, E_Bi_dict = unpickler('E')
    F_Uni_dict, F_Bi_dict = unpickler('F')
    I_Uni_dict, I_Bi_dict = unpickler('I')

    # read in test file
    with open('LangId.test', 'r') as f:
        test_file = f.readlines()
    f.close()

    # read in solutions file
    with open('LangId.sol', 'r') as f:
        solutions_file = f.readlines()
    f.close()

    #For each line in the test file, calculate a probability for each language
    #and write the language with the highest probability to a file
    V = len(E_Uni_dict) + len(F_Uni_dict) + len(I_Uni_dict)

    for each_line in test_file:
        E_prob = calc_prob(each_line, E_Uni_dict, E_Bi_dict, V)
        F_prob = calc_prob(each_line, F_Uni_dict, F_Bi_dict, V)
        I_prob = calc_prob(each_line, I_Uni_dict, I_Bi_dict, V)

        #if english has highest probability
```

```
        if E_prob > F_prob and E_prob > I_prob:
            with open('highest_prob.txt', 'a') as f:
                f.write("English\n")
            f.close()

        #if french has highest probability
        elif F_prob > E_prob and F_prob > I_prob:
            with open('highest_prob.txt', 'a') as f:
                f.write("French\n")
            f.close()

        #if italian has highest probability
        else:
            with open('highest_prob.txt', 'a') as f:
                f.write("Italian\n")
            f.close()

    # read in from predictions file
    with open('highest_prob.txt', 'r') as f:
        highest_prob = f.readlines()
    f.close()

    # keep track of line numbers that are incorrectly classified
    inc_line_num = []



    #calculate accuracy of predictions
    correct_prediction = 0

    for each_line in range(1,len(solutions_file)):
        if highest_prob[each_line] == solutions_file[each_line]:
            correct_prediction += 1
        else:
            inc_line_num.append(each_line + 1)     # +1 because list index will start a

    # Compute and output your accuracy as the percentage of correctly classified instan
    accuracy = correct_prediction / len(highest_prob)

    # output your accuracy, as well as the line numbers of the incorrectly classified i
    print("Accuracy: ", accuracy)
    print("Incorrect Classification line numbers: ", inc_line_num)
```

In [10]:

```
if __name__ == "__main__":
    main()
```

```
Accuracy:  0.0
Incorrect Classification line numbers:  [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81,
82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 1
03, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120,
121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 13
8, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,
156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 17
3, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190,
191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 20
```

```
8, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225,
226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 24
3, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260,
261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 27
8, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295,
296, 297, 298, 299, 300]
```

In [ ]: