HG.

Output sample

```
Which positions would you like to see?
Start: 1
End: 3
#1 most common word: [the] with 1,372,271 occurrences
#2 most common word: [of] with 597,086 occurrences
#3 most common word: [to] with 568,300 occurrences

Which positions would you like to see?
Start: 7
End: 7
#7 most common word: [s] with 264,098 occurrences

Which positions would you like to see?
Start: 30
End: 38
#30 most common word: [were] with 68,094 occurrences
#31 most common word: [they] with 67,772 occurrences
#32 most common word: [year] with 64,881 occurrences
#33 most common word: [not] with 62,004 occurrences
#34 most common word: [would] with 59,572 occurrences
#35 most common word: [this] with 58,737 occurrences
#36 most common word: [mr] with 55,748 occurrences
#37 most common word: [had] with 55,690 occurrences
#38 most common word: [t] with 55,593 occurrences

Which positions would you like to see?
```

Let user ask questions repeatedly.

I use a hash symbol table with linear probing. Linear probing handles collisions, so if two keys were calculated to have the same hash the later key's hash will be recalculated to something else so that collisions won't occur. This means that every key-value pair will have it's own index in the symbol table, making it easy to sort with, for example, quick sort.

I use hash symbol table to be able to search for keys in constant time.

When searching in table, we can access keys with the hash values instead of having to iterate through all hashes and comparing keys.

We know from exercise 3 that the hash values are distributed very evenly to the keys if we set the interval to 0 to the number of keys (distinct words). This means that when we want to insert keys with occupied hash values, we only need to

recalculate for a very few amount keys. Same goes for when searching for already existing key.