https://github.com/RohinBhargava/Autofill-Simulator

Overall structure of program with diagram.

1. Parsing routines.
2. Hashing and consequences.
3. Structures and data storage.
4. Getting data for output.
5. Improvements.

1. The parser takes one sentence at a time of the inputs and tokenizes them for input. Then we initialize a new word_tree using the hashed first word and then add the words and sort them as they're entered to create a weighted tree. If there is a collision after hashing, there is a built in check for that.
2. The hash function is a 32-bit FNV-1, which multiplies an input char by char by a prime at each iteration and uses a special sum of bitshifts to create unique buckets in the range of 2^32.
3. The structures are as follows. Everything should be intuitive except for "left", which is the case for collisions.

```
typedef struct word {
   char *name;
   int score;
} word;

typedef struct word_tree {
   struct word *name;
   struct word_tree *left;
   struct word_tree **children;
   int tree_count;
} word_tree;

typedef struct word_hash {
   struct word_tree **hash;
} word_hash;
```

4. We follow a simple tree traverse that concatenates words as we go along a tree, first returning the left most side of our hash (which is the highest relevance).
5. Better parsing routines! Perhaps I should try to implement something in yacc/lex to include more parsing possibilities. Also, I should constrain top hits, probably with a global counter. I want to make sure this works on many people's machines. Finally, I want a faster free method – perhaps keeping a list of first words will help with this.