* We will query for sequences from a genome file
* Implementation
  + Index the genome file with a series of k-mers
  + K-mer is a sequence of k letters (there may be repeated letters) from the DNA alphabet (A, T, C, G) where k is an integer
  + A given k-mer can appear many times within the genome
  + The k-mers will be the keys for the hash table
  + Hash table will be implemented as an array or a vector at the top level
* Hash function will map the k-mer to a position in the table
* Choose an appropriate structure to store the k-mers and genomic locations of the k-mers (the positions where they are found in the genomic sequence)
* Iterate through the genome sequence with a series of overlapping windows of length *k*, calculating the index into the hash table using the hash function
* Store the k-mer and its genomic location in the table
* When iterating through the genome sequence, the first k-mer is the genome sequence from 0 to k-1, the second is the genome sequence from 1 to k, etc.
* Process queries of varying length and allow for mismatches between the genome and query string
* Use the first k letters of the query string as a seed 🡪 it’s important that searching the database for the initial seed be efficient
* If the seed can be found in the table, the program should try to extend the match by adding letters from the indexed genomic position until the full query is matched, or the allowed number of mismatches is exceeded
* We require that the seed be an exact match 🡪 the mismatches may occur anywhere after the seed in the match string
* You can choose your own hash function 🡪 it must be fast, O(1), and provide a random, uniform distribution of keys throughout the table
  + You can use one of the hash functions used in lecture, one found on the internet, or one of your own devising
    - If you’re using a hash function from the Internet, you must provide a URL in your README and include the source code with your submission
    - If the downloaded file requires a copyright notice, you MUST include that notice
    - Be sure to observe any copyright restrictions on the use of your code
    - In your README file, describe your hash function and table implementation
* A typical k-mer will be found in several locations in the genome 🡪 your hash implementation should enable efficient retrieval of the multiple locations where the k-mer is found
* To store the k-mers and their genomic positions in the hash table, once the table index of the kmer key has been found, you may use any of the data structures we have covered so far in class
* To handle collisions, use one of the open addressing methods described in lecture (linear probing, quadratic probing, or secondary hashing) 🡪 linear probing is the simplest of these three methods
* You may not use std::hash, std::unordered\_map, std::unordered\_set, std::map, or similar STL functions/containers
* When implementing the hash table, set the initial size of the table 🡪 as you enter data in the table, calculate the occupancy ratio
  + occupancy = number of unique key entries/table size
  + When the occupancy > than some fixed level, double the size of the table and rehash the data
  + Describe your re-sizing method in the hash table section of your README
* Input and output functionality (redirect cin and cout)
  + Input file
    - genome filename
      * Read a genome sequence from file name, genome file consists of lines of DNA characters
    - table\_size N
      * Optional command, N is an integer (initial hash table size), default is 100
    - occupancy f
      * Optional command (f is a float 🡪 when occupancy is above this level, the table should be resized), default value = 0.5
    - kmer k
      * Genome should be indexed with kmers of length k
    - query m query\_string
      * Search the genome for a match to query\_string allowing for m mismatches
    - quit
      * Exit the program
  + Output file
    - The program will report the query
    - If matches are found, the genome position (positions start at 0) of the match, the number of mismatches between the genome and the query string, and the genome sequence matching the query
    - If a match can’t be found, the program will report “No Match”
* Extra credit
  + Add a new command to implement the database using one of the other structures that we have covered so far in the course: vectors, lists, arrays, etc.
  + Compare the performance of your alternative method to the homework method by making a table of run times for each of the genomes and query sets provided with the homework and compare the times to build the index and the times to process the queries
  + Document any new commands you have added in your README