This part of the project required the program to instead predict the user's query and print out the relevant parts and items as a result.

I decided to make a Query class with the appropriate header and implementation files that would be used mainly to handle the user's query. I utilized the splitQuery method to split the user's query into a vector of strings that all contained a substring word. I used a space delimiter and stringstream to handle this, as well as a toLower method so the cases would not be a problem. Although it does not handle multiple spaces well, it is able to effectively split a string sentence to its individual words.

One thing I wanted to do was create a vector of strings for each of the intents, and then have a for loop loop through each index of the vector, split the string at that index, and then compare it to the split user query. However, I found that this was too complex and that a lot of the possible intents I had written were all too similar to each other, with only a few words separating them. At that point, I felt that the regexes I had created for each intent likely handled matching the user's query to the specific part or item they requested, so I decided to forgo this. It also felt like trying to instead just have one specific question for the intent that the query would match to would be too narrow as well. This is a big part of this part's program that I am not quite sure how to handle at the moment without making the program much more complex than my current abilities allow.

Instead, I decided to make a matchCompany method for the two companies: Netflix and Paramount, that would be used first in my main .cpp program. This function first calls the splitQuery function, and then loops through each word in the user query, attempting to find a match to either "netflix" or "paramount". If it finds a match, it increments the specific netflixScore or paramountScore. After it finishes looping through the query, it compares the

netflixScore to the paramountScore, and returns the string name of the file whose score was higher than the others. This is to handle any situations where the names of the two companies appear multiple times. The program will simply just choose the company that appears most in the query if such a situation arises.

Overall, I think there could definitely be something I was missing with mapping user query to intent. I felt like the regex_search functions that I used handled most of the user intent, as the regexes were decently complex enough to consider common misspellings or omissions of words, and any attempt to try and match user query to a specific intent that would be better would likely require comparing the chars of each individual word and matching that way or something more complex.