MustangWiki Search Engine

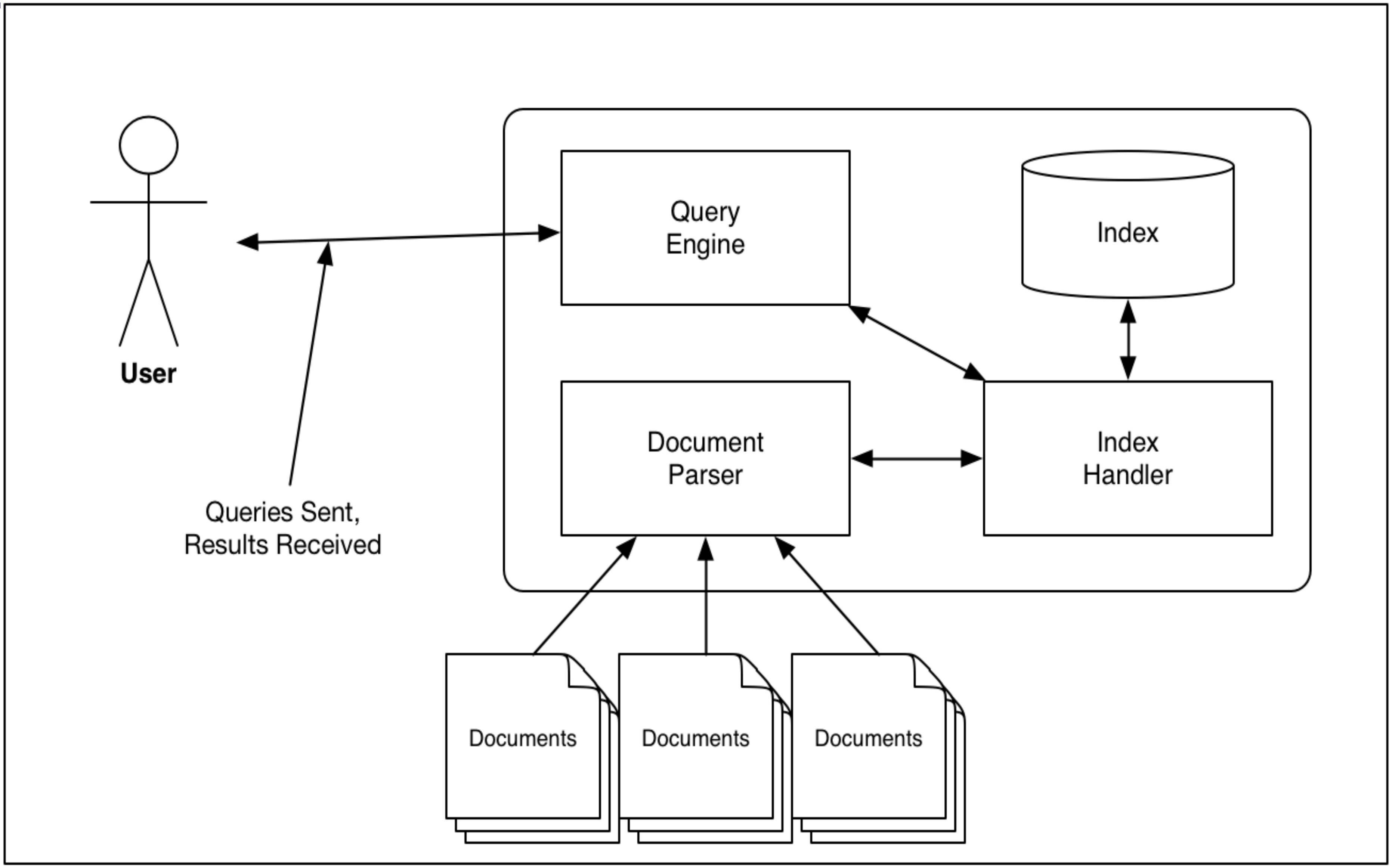
CSE 2341 - Data Structures

Semester Project Overview

Scenario

- MustangWiki's search engine is broken!
- MustangWiki is a collaborative, open place where students, faculty, staff and the community can create online, freely-accessible content.
- Please help MustangWiki get its search abilities back by implementing a fast search engine for all of the books in the collection.

System Architecture



Sample Document

```
<page>
    <title>Human Anatomy/Osteology/Axialskeleton</title>
    <ns>0</ns>
    <id>181313</id>
    <revision>
      <id>1481605</id>
      <parentid>1379871</parentid>
      <timestamp>2009-04-26T02:03:12Z</timestamp>
      <contributor>
        <username>Adrignola</username>
        <id>169232</id>
      </contributor>
      <minor />
      <comment>+Category</comment>
      <sha1>hvxozde19haz4yhwj73ez82tf2bocbz</sha1>
      <text xml:space="preserve">[[Image:Axial_skeleton_diagram.svg|thumb|240px|right|
Diagram of the axial skeleton]]

The Axial Skeleton is a division of the human skeleton and is named because it makes up
the longitudinal ''axis'' of the body. It consists of the skull, hyoid bone, vertebral
column, sternum and ribs. It is widely accepted to be made up of 80 bones, although
this number varies from individual to individual.

[[Category:{{FULLBOOKNAME}}|{{FULLCHAPTERNAME}}]]</text>
    </revision>
  </page>
```

Inverted File Index

- Data structure for maintaining terms and a list of documents in which those terms appear.

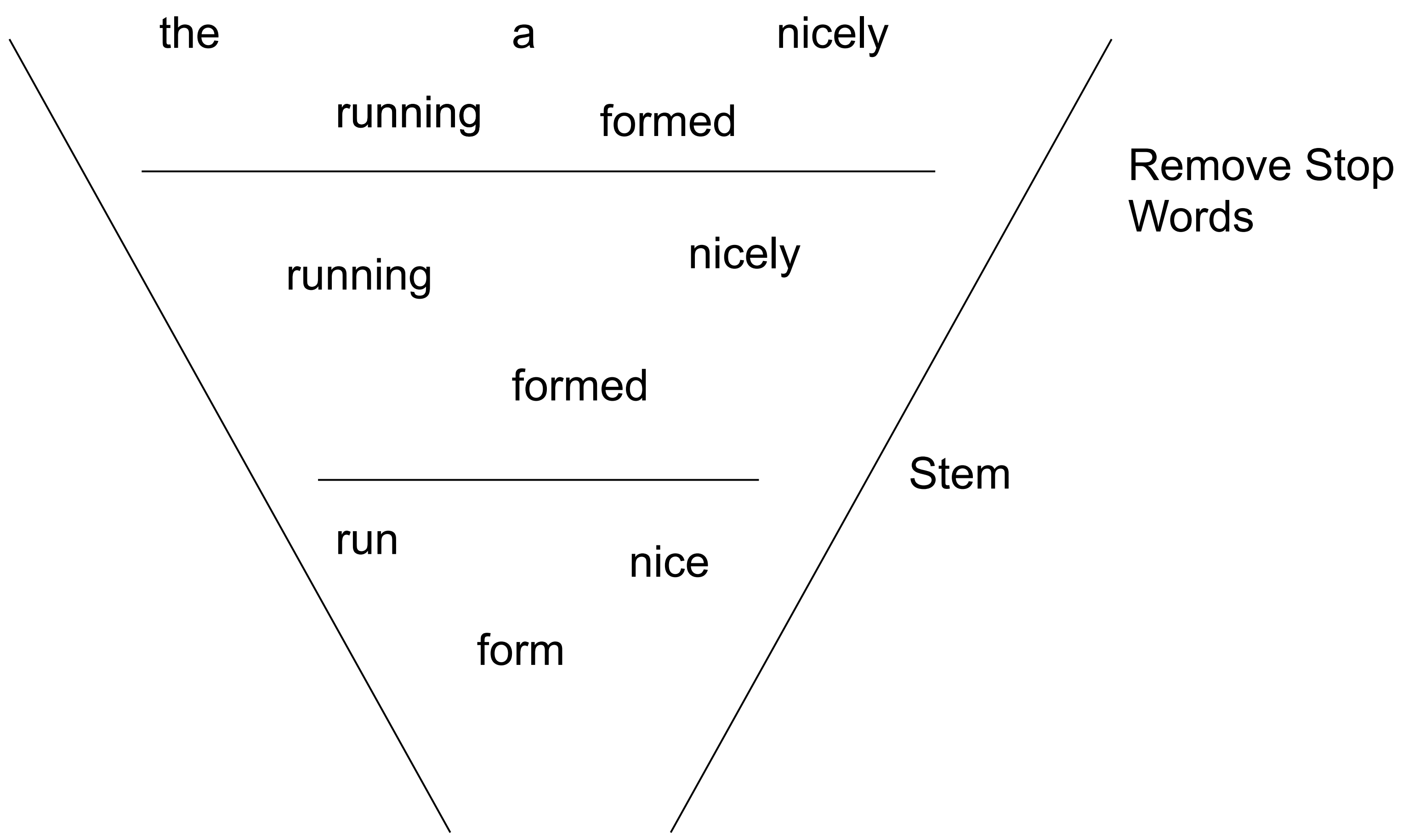
<u>Documents</u>:

```
d1 = computer network security
d2 = network cryptography
d3 = database security
```

Index:

```
computer = d1
network = d1, d2
security = d1, d3
cryptography = d2
database = d3
```

Document Parser



Query Processor

- Will handle simple prefix Boolean queries
  - no nesting
  - Either AND or OR
  - May include NOT (all NOTs will be trailing)
- Examples:
  - Seattle
  - Seattle NOT Boston
  - AND book food bank
  - OR Boston Seattle
  - AND Book Boston NOT Seattle

Index Handler

- Creation and maintenance of *inverted file index*.
- Search and returning documents containing a specific word (query term)
- May maintain other information such as frequency of word appearance.

Inverted File Index Implementation

- You'll implement at least two underlying data structures to maintain the inverted file index
  - AVL Tree
  - Hash Table
- Classes that will be used to store index should implement the same interface.
  - may be adapters to AVL or hash table.
- Index should be persistent.
  - When program starts, user should have the option of importing index into AVL or Hash table.

Ranking the Results

- Ranking will be done using Term Frequency/Inverse Document Frequency (TF/IDF) statistic.
- General Idea:
  - If a word appears in a document frequently, but appears rarely in other documents from the same corpus, it is important result for the query.
  - If a word appears in a document but also appears in many other documents, it is less important.

User Interface

- 3 modes:
  - maintenance mode
    - add to the index; clear the index
  - interactive mode
    - user can enter a query,
    - return ranked results
    - allow user to choose a page to view
  - stress test mode
    - allow user to submit a command file.
    - Commands are created by each group and documented

Project Mechanics

- Can be done individually or in in teams of 2 or 3 students (max)
  - Teams of 3 have to implement basic date range queries and user/writer queries
- Must use OOP
- May use as much of c++ std lib as you'd like
- Code base should be properly documented and formatted.

About the Dataset

- Wikibooks export file
  - ~171,000 individual pages (<page></page>)
  - >700 MB of raw
  - XML format
  - Not pretty
  - May contain non-ascii characters
- *Goal: Parse and index in under 2 minutes!*

Suggestions

- This is a ***gigantic*** project
  - May contain > 20 classes all working together
- You have to start NOW!
- Spend up-front time on ***design***!
- Implement one piece at a time.
  - Once that works and is tested, back it up.
  - Think about versioning of software.
    - 0.1, 0.2, 0.3...
    - Make an initial list of goals for each version and when they should be done.