# Information Retrieval Course
# CSE 435/535 - Fall 2010
# Project 1
# Document tokenizer in C++ for Wikipedia markups

**Project Description** An IR Engine should include at least the following major components: Text parser, Indexer and Retrieval System. Your first project is to build the text parser which will be used by subsequent projects. It must be coded in C++. One of the purposes of this project is to let you get familiar with C++ STL (Standard Template Library) and also process documents that have wikipedia style markups.

*Due Date* **Online submission through UBLearns' Digital Dropbox by** $29^{th}$ **September, 2010 11:59 P.M.**

## 1 Parsing/tokenizing plain english text

**In memory tokenization and dictionary creation**

**TODO**

- **Tokenization:** Read documents into memory, tokenize to separate words/tokens in the documents, process the tokens and store the "terms" in a suitable STL container. Tokenization rules will be discussed in class and recitation.
- **Term Dictionary:** Build a "dictionary," which assigns each processed word/token to a numerical ID and keeps this correspondence information. Each raw token is first processed into a corresponding term (for e.g. run→run, running→run, etc.) and each unique term is assigned to a unique global numerical ID.

  The dictionary should be implemented in a way such that element lookups are very fast. We will assume that the IDs here will be of data type *size_t*. In real world scenarios it makes sense to assign IDs that are represented as "Big Integers" possibly converted to hexadecimal strings.
- **File Dictionary:** You also need to keep a dictionary to map each document name to a unique numerical ID also of type *size_t*
- **Term Count Dictionary:** You also need to keep a dictionary to map each unique numerical term ID to its count in the corpus

NOTE: you don't need to invent a new STL container for this class project. You are also free to use any open source data structures like Google Sparse hash tables `http://code.google.com/p/google-sparsehash/` or SGI stl hash_map `http://www.sgi.com/tech/stl/`. Hash maps are typically not ordered and if you want ordering, as is often the case when you want to view the map dumped in a text file, feel free to use stl maps which are implemented using red-black trees having a lookup complexity of $\mathcal{O}(\log n)$ with $n$ being the number of elements in the map.
Document preprocessing steps typically include:

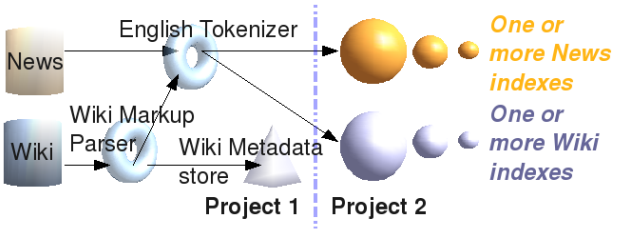


Figure 1: Schematic diagram of project 1

- Tokenization to handle numbers, **hyphens**, punctuation marks, the case of letters (upper/lower), removal of Wiki markups
- Elimination of stopwords
- Stemming of the remaining words

The dictionary will be a simple text file with each line formatted as <Key Value> without the <'s and the >'s. *Note the space.* For example, <Key Value> could be <TermID TermString> etc.

## 2 Parsing/tokenizing Wikipedia text

### In memory parsing

In this step the Wiki markup files will be parsed to grab some document metadata and the original text will then be fed into the tokenizer and the dictionary generator that you have created in the previous step. (cf. Figure 1).

Although we will be parsing a smaller subset of the Wiki markup tags, for detailed descriptions of wiki markup tags, please visit `http://en.wikipedia.org/wiki/Help:Wiki_markup`. For our purposes, we will be interested in extracting "Infoboxes" and "Taxoboxes", Section and Subsection title strings and Internal Wiki links. Refer to Table 2 for a quick view. The markup parser will help in creating text files corresponding to the original wiki files that stores the following meta information:

- Infoboxes or Taxoboxes
- Section and Subsection headings and a short chunk of text thereafter
- Internal wiki links

**TODO**

- **Simple Parsing:** Generate a unique document ID (of type size_t) for each wiki document.
  For each wiki document generate a text file named <wiki document ID>_semwiki_meta.txt that looks like

```
<#INFOBOX>
.... dump infobox contents here
<#SECTIONS>
heading_text_1 $ 20 words or less following the heading text 1
heading_text_2 $ 20 words or less following the heading text 2
....
heading_text_n $ 20 words or less following the heading text n
<#LINKS>
link_1 $ link_2 $ .... $ link_K
<#CATEGORY>
category_1 $ category_2 $ .... $ category_M
```

- **File Dictionary and store:** Generate a unique document ID (of type size_t) for each semantic wiki meta document and add it to the main file dictionary. Also write each meta file to a directory named "SemWiki" with the filename given above.
- **Link Dictionary** You also need to keep a dictionary to map each unique internal wiki link to a corresponding unique global link ID

**Note:** *While parsing pay special attention to tags for Infoboxes and Taxoboxes and the tags mentioned in Table 2.* Also ignore everything after the *last* "[[Category:xxx yyyy]]" string encountered in the Wiki markup file.

**A Word About Parsing, Tokenization and Dictionary Creation**

In general, when a text document is read from disk, it will be stored internally as a list of space separated tokens (not terms which are processed tokens). Thus for a document that is not a wiki document, you might invoke a simple tokenizer that processes the token list for creating the dictionaries. For a document that is a wiki document, it is first parsed to grab semantic metadata for the wikipedia article and then its token list could be processed through another tokenizer that is derived from the base tokenizer for the remaining wiki markup elimination.

In real life, when a document is crawled, there is some mechanism to identify the major class of the document like whether it is a news document, a wiki article, a blog article, a page generated by submitting a query to a shopping website etc. For our purposes, we will store two types of documents in two directories named "News" and "Wiki" (see section 3). All files under the same directory contains files belonging to the same category. Thus the directory name will give you enough hint as to what kind of document it is.

Although there are two different folders here, there will only one term dictionary, one file dictionary and one term count dictionary for all the text files. Note that the meta text files will not be indexed in the usual sense of inverted indexes but will be used in a clever fashion in projects 2 and 3.

For writing the report, you will need to write some statistics found from the corpus that you have parsed and/or tokenized. This will contain information on:

- How many terms were there in the corpus
- How many raw files were there in the corpus all together
- How many semantic wiki metafiles were generated all together
- What are the term counts of the terms that mark the top 95% and the bottom 5% of all the terms in the vocabulary w.r.t term counts.

Also include as a table with two columns:

- A set of the "infobox" or "taxobox" labels (the string that appears to the left hand side of '=' ) on the first column and possible expanded text on the second column.

## 3 Data

The data will be organized in two folders:

- **News**: Documents in this folder have originated from news sources and had been used in competitions like TREC.
- **Wiki**: Documents in this folder are extracted from Wikipedia. These documents have been/are being crawled and fetched beginning September 2, 2010 and onwards. The Wiki markups are extracted from the <text> portion found in the XML feed from `http://en.wikipedia.org/wiki/Special:Export/<Wikipedia_article_name>`

At the beginning of project 1, you will be given samples from these two sources just to get you started on tokenization and dictionary creation. More files will be supplied as you move along towards the end of Project 1. In project 2, you will be given the full data sets to index. Note that the News dataset will primarily be used in homeworks.

## 4 Important Information

Please ask at least one your project members to attend recitation for information on which tools and IDE to use for developing your programs. I personally use Eclipse CDT on Ubuntu. If you use Eclipse, the makefiles will automatically be generated for you. If you prefer to use windows, install Cygwin (`http://www.cygwin.com/`) and then extract Eclipse CDT for windows. In the latter case, the Cygwin GCC toolchain should automatically be selected for compilation. All programs must be coded in C++ and **should not be** in only 1 or 2 (one header and one source) file(s). Points will be allocated for clean and modular design. Your code should be easily modifiable and extensible so that you can build on this codebase for projects 2 and 3 .

## 5 Submission

Submit a tar or zip file named <your-team-name>.tar or <your-team-name>.zip using the Digital Dropbox in UBLearns. The contents of the compressed file is as follows:

−− <your-team-name>.tar
|+ document_processor.tar (All source and make files + readme files)
|+ report.pdf (A report showing snapshots of the dictionaries, statistics about the dictionary contents, the table containing infobox and/or taxobox labels and values and a sample snapshot of any two semantic metafile generated as outputs)

Your program should run as <your-program-name> <base-document-directory> index
"index" is the name of the output-directory that contains all the dictionary files.

Make sure all your programs at least compile on at at least one CSE machine: (e.g. metallica.cse.buffalo.edu [running Linux]) Please read the contents of `https://wiki.cse.buffalo.edu/services/content/student-servers` to find out which system to use. For example for test purposes on smaller document collections you can use timberlake. If there is a demo, then during the day of the demo, the TAs will download your most recent tar files (upto the submission date) from UBLearns and run it.

| | |
|---|---|
| Infobox | {{Infobox Tennis player<br>\|playername=<br>\|nickname= ''Boom Boom''<br /> ''The Lion of Leimen''<br>\|country= {{flagicon\|Germany}}[[West Germany]]<br>(19831990)<br />{{GER}} (from 1990)<br>\|residence= [[Schwyz]], {{CHE}}<br>\|datebirth= {{birth date and age\|df=yes\|1967\|11\|22}}<br>\|placebirth= [[Leimen (Baden)\|Leimen]],<br>{{flagicon\|Germany}}[[West Germany]]<br>\|height= {{height\|m=1.90}}<br>\|weight= {{convert\|85\|kg\|lb st\|abbr=on}}<br>\|turnedpro= 1984<br>\|retired= June 30, 1999<br>\|plays= right-handed; one-handed backhand<br>\|careerprizemoney= US $25,080,956<br /><br>* [[ATP_Tour_records#Earnings\|5th<br>All-time leader in earnings]]<br>\| tennishofyear = 2003<br>\| tennishofid = boris-becker<br>\|highestdoublesranking= 6 (22 September 1986)<br>}} |
| Taxobox | {{Taxobox<br>\| name = Cucumber<br>\| regnum = [[Plant]]ae<br>\| divisio = [[Flowering plant\|Magnoliophyta]]<br>\| classis = [[Magnoliopsida]]<br>\| ordo = [[Cucurbitales]]<br>\| familia = [[Cucurbitaceae]]<br>\| genus = ''[[Cucumis]]''<br>\| species = '''''C. sativus'''''<br>\| binomial_authority = [[Carolus Linnaeus\|L.]]<br>}} |
| Sections and subsections | ==Section headings==<br>''Headings'' organize your writing into sections.<br>The Wiki software can automatically generate<br>a table of contents from them.<br>Start with 2 'equals' characters.<br>===Subsection===<br><br>Using more 'equals' characters creates a subsection.<br><br>====A smaller subsection====<br>Don't skip levels, like from two to four 'equals' characters. |
| Links | London has [[public transport]]<br>New York also has [[public transport\|public transportation]]<br>[[kingdom (biology)\|]]<br>[[Wikipedia:Village pump\|]]<br>[[Wikipedia:Manual of Style (headings)\|]]<br>A [[micro-]]<nowiki>second</nowiki> |

Table 1: Quick view of relevant wiki markups