Conlang Compiler Design Document

**Premise:** Create a program which assists in the creation of constructed languages.

# Components:

* Dictionary Component
* Context Free Language Component
* Translator Component

# Interface Options

* **Main Menu**
  + Create New Language
    - Name the Language
    - Choose Language Etymology (monosyllabic or otherwise)
    - Type language description
    - **Progress to [LANGUAGE MENU]**
  + Load Existing Language from File
    - Type in filename
      * Check if file exists, if not allow them to reenter it or quit.
        + **Progress to [LANGUAGE MENU]**
      * Back
        + **Progress to [MAIN MENU]**
  + Quit
    - **[EXIT PROGRAM]**
* **Language Menu**
  + View language statistics
    - Print Language Statistics which include:
      * Language Name
      * Language Etymology
      * Language description
      * Language word count
      * Rules in Grammar
  + Grammar Options
    - View Word Types
    - View Productions
    - Add Word Type
    - Remove Word Type
    - Add Production
    - Remove Production
  + Dictionary Options
    - Search Language
      * View Word
      * View Meaning
      * List Words
      * List Words by Type
      * Back
        + **Progress to [LANGUAGE MENU]**
    - Add to Language
      * Add a word in entirety
      * Randomly Generate a word from stems and assign meaning
      * Randomly Generate words from stems and assign meaning and type from a list of English words.
      * Back
        + **Progress to [LANGUAGE MENU]**
    - Remove from Language
      * Find word and remove it from the language
      * Remove a stem from the language (tells you how many words you’d get rid of)
      * Back
        + **Progress to [LANGUAGE MENU]**
    - Sentence Functions
      * Make a random sentence
        + **Progress to [LANGUAGE MENU]**
    - Quit

# General

In creating the Conlang we are trying to make a tool which can be used to effectively create constructed languages, which really means we’re designing something which can store and simulate languages. Without a background in linguistics, this is going to be very difficult, and I should probably do my share of research on that massive topic somewhere in the production of this thing. As it stands, I’m more willing to get a skeleton down before hand, and then dive into details, just to know whether or not this thing will be feasible at all. The basic structure is as follows:

* I need something to deal with grammar, i.e the context free grammar module,
* I need something to deal with storing all of the words which will be made by the language, I want to get an sql database for them but I’m not sure that’s going to work, so I’ll hold on hope for that.

In terms of standard interface we’re going to need the ability to:

* Make a new language
* Load an existing language which should be a folder which will store various files and eventually link to an sql database with various tables. These tables are going to be the different kinds of words, and the grammar, as well as special rules for them.
* The first file to be read will be the grammar, which will lay out what kinds of words exist in the language and what kinds of relations they have. Then, based on that list of word types, the program will read the various other files in the language.
* The second files will be from the words of the language.
* The user will be able to construct new words in the program and assign meanings to them, as well as check whether certain sentences would be acceptable in the language.

**Grammar Specific**

Two grammar files, SentenceConstructionKit, and WordConstructionKit, both are context free grammars. The first deals with the rules of constructing sentences out of existing words according to the rules of the language and the second deals with constructing the words themselves, i.e, <prefix> <noun>, <noun> <noun>, <prefix> <verb>, <verb> <modifier>. The WordConstructionKit is what’s used when randomly making new words to give meanings to.

What I actually need here it to write out context free languages. Which is a lot but you know what I mean.

**Word Types**

**Format:**

<begin preterminal>

[List of wordtypes, each separated from each other by a comma and space, no period at end]

<end preterminal>

<begin nonterminal>

[List of sentence helper nonterminal types, each separated by a comma, no period at the end]

<end nonterminal>

**Example:**

<begin preterminal>

noun, verb, adjective, adverb, pronoun, preposition, conjunction, determiner, exclamation, prefix, flourish, number, modifier

<end preterminal>

<begin nonterminal>

complex noun, complex verb, complex adjective, list noun, list adjective

<end nonterminal>

**Sample Sentence Construction Kit**

<begin construction>

complex noun – noun | preposition noun | preposition noun | preposition noun complex adjective

complex verb – verb | noun verb | verb adverb | adverb verb

complex adjective – adjective | list adjective

<end construction>

**Sample Word Construction Kit**

Each indent is 4 spaces. Each element of the list is a construction of preterminal symbols which can be put together to legally form a new word, such as “adjective noun” which means words like “darkwood” are possible in the language.

<begin construction>

adjective noun

noun verb

verb noun

prefix noun

noun suffix

verb modifier

noun modifier

<end construction>

Each one of these things will populate a list of enums.

**Class Creation Flow**

* First we make Scrivener which checks whether the files we need exist
* Then we make Grammar and call Scrivener to get us the info we need to make the types of Symbols. We also make a vector of symbols for each symbol type we encounter.
* We then make codex and assign grammar and scrivener to it. We then use scrivener to read the word files, and using grammar, we add any word with the appropriate symbol to the list of them in grammar.
* Then in codex we can make functions which call grammar to construct