1 Introduction

Creating text files is a daily occurrence for nearly all students nearly all working professionals. Currently, the two main options we use for these tasks, LaTeXand Microsoft Word, both provide less than optimal user experiences. LaTeX, for example, is far more powerful than the vast majority of users need it to be, and the heavy emphasis on formatting makes it difficult for an unskilled user to create the document they have in mind. Microsoft Word is on the other end of the spectrum. What you see is what you get, but the feature set is lacking, the editor itself runs slowly on older machines, and the .docx format doesn't play nice with other formats.

The language I have created, LiTeX, seeks to solve these shortcomings while emphasizing ease of use, lightweight editors, and portability. This problem needs its own programming language to extend the functionality of lightweight text editors while avoiding complex syntax present in other typesetting software. As a proof of concept, version V1.0 of LiTeX will convert basic text into HTML (mostly because the XML .docx formatting manual is 5207 pages long) with future development focusing on other file formats.

2 Design Principles

The underlying design principle present in LiTeX is ease of use. There should be as close to zero learning curve as possible when someone picks up the software. There should also be very little setup time between when you open the software and when you can start writing freely.

3 Example Programs

Example 1:
This is a LiTeX program with simple text. All alphanumeric characters are allowed.
Linebreaks are ported 1:1

As well as paragraphs.

This program, which can be run with

```
dotnet run example-1.txt example-1.html
```

will ouput an HTML file with standard font selection and color, with the inputted text exactly as it appears in the input file.

Example 2:
This is a LiTeX program with modifiers. +Bold+, *italicized* and _underlined_modifiers are all allowed and can be +*_combined_*+.
~1~Headers are allowed using tildes.~1~
=30=As well as font sizes.=30=
This program, which can be run with

```
dotnet run modifiers.txt output.html
```

will output an HTML file with the modifiers given applied to the HTML output.

Example 3:
MACRO spam := +*
This is a LiTeX program with macros.

#spam#This text will be both bold and italic#spam#

This program, which can be run with

```
dotnet run example-3.txt example-3.html
```

will output an HTML file with modifiers applied to all text inside of that macro.

4 Language Concepts

The core concept that a user needs to understand to use LiTeX is the distinction between modifiers and words. Words can be written at any point in the program, with the exception of inside modifier flags and after macro definitions. For correct programs, modifiers must always occur in pairs to correspond to the HTML tags they will create. Modifiers can always be compounded upon each other to wrap basic text, the primitive in the language, both in the program and as part of a macro.

5 Syntax

The syntax will be incredibly easy to execute, especially for non-technical people. All words will be written as the user wants them to appear in the PDF, including white spaces and new lines. Modifiers will correspond to particular special characters. The words that appear within the modifier characters will have that particular command applied. Specific series of commands can be saved for future use with the MACRO keyword followed by a variable name, followed by the assignment symbol and the modifiers you want present in the macro. Backus-Naur form is given below.

⟨*expr*⟩ ::= ⟨*item*⟩+

⟨*item*⟩ ::= ⟨*word*⟩ | ⟨*modifier*⟩ | ⟨*macro*⟩ | ⟨*macroPhrase*⟩

⟨*word*⟩ ::= (a-Z,1-9, ⟨*punctuation*⟩) +

⟨*modifier*⟩ ::= ⟨*ital*⟩ | ⟨*bold*⟩ | ⟨*underline*⟩ | ⟨*header*⟩ | ⟨*size*⟩ | ⟨*paragraph*⟩ | ⟨*newline*⟩

⟨*paragraph*⟩ ::= '/n'

⟨*newline*⟩ ::= '/r'

⟨*ital*⟩ ::= '*'

⟨*bold*⟩ ::= '+'

⟨*underline*⟩ ::= '_'

⟨*header*⟩ ::= '~1~' + ⟨*number*⟩ + '~1~'

⟨*size*⟩ ::= '=' + ⟨*number*⟩ + '='

⟨*macro*⟩ ::= 'MACRO' + ⟨*varname*⟩ + ':=' ⟨*modifier*⟩+

⟨*varname*⟩ ::= (a-Z)+

⟨*macroPhrase*⟩ ::= '#' + ⟨*varname*⟩ + '#'

6 Semantics

The most basic statement in LiTeX is a word, which is any collection of letters, numbers, standard punctuation and whitespace. This will be transfered 1:1 to the HTML output. All modifiers serve as lightweight markers for the start of a particular text modification These can be combined in sequence with one another to create different outputs. Of these modifiers, size and header are unique in that they create a paragraph element simply by existing due to the nature of HTML. This is not ideal for the usability of the language, but it is necessary. The paragraph and newline modifier are created naturally when the user inputs a carriage return or new line. This preserves the ”what you see is what you get” paradigm that LiTeX emphasizes. Macros are definitions of modifier groups that can be saved and used multiple times throughout the program. They are parsed and stored, but not exported into the HTML document. For best results, these macros should be defined in the first lines of the LiTeX program. If not, there is a one-line spacing difference between the .txt file and the outputted HTML. MacroPhrases use a stored macro as a modifier to effect the text between them. This syntax has been designed to match closely the natural manner of writing to be as unobtrusive to the process as possible.

7 Remaining Work

In the future, I would like to add more functionality to LiTeX. The first item on this list is support for left/right margin adjustment. This could be done by counting the number of spaces/tabs at the start of each line and adjusting the margin of each HTML line accordingly. This would make the language more helpful for rapid prototyping of web content. Additionally, at the start of this project I wanted to add simply math expression support so users wouldn’t have to alt-tab to google calculator or pull out their phones to add or multiply something. This would be a great feature for science reports and other more technical papers. This feature proved very difficult to implement because of the base of the language being all in strings.