Markdown Converter

Elijah Augustin

CSCI 49102

The purpose of this project was to implement a simple Markdown to HTML converter in the OCAML programming language using some of the principles of functional programming such as immutable data, using functions for the core design of the project and any side effects. The Markdown specifications that I used to implement my converter was a combination of the Github Flavored Markdown spec and the Common Mark spec.

Before the process of the conversion, I define a type of Markdown Classification which can consists of many of the typical Markdown symbols such as Heading, Paragraph, Blockquote, List or Empty. There is also a type of Unknown which is a placeholder for types that the parser cannot classify as of now. When the program is run, first markdown file is read into a list of strings. Next I map a conversion function over the list of strings to produce a list consisting of Markdown Conversion types. The Str library in Ocaml which hosts many functions for using regular expression to search strings is used in the process to map a string into a Conversion type. However the power of regular expressions in the Str library is limited and not to the same level as normal regular expression in other languages such as using certain special symbols to indicate how many character to check for. Due to the limitations of the Str library, certain Markdown types like bold/italic and tables are very difficult to implement and are not in this current version of my converter.

Since we are converting the markdown file line by line, the question arises, “How do we convert types that span multiple lines?”. The answer to that question is that currently the converter has 3 types that is groups together BlockquoteItem, UnOrderedListItem, and OrderedListItem. When the converter first encounters any of these types, it passes the item and the rest of the conversion list to a helper function which checks the next occurring lines. If those lines are of the same type, then they are grouped together into a single data type and inserted in the conversion list. After we have the list of conversion types, another function is mapped over the list which produces a html string based off of what the conversion type is being passed to it. The predefined regular expression for each markdown type is used in conjunction with substitution and replacement methods from the Str library to convert those symbols into html tags that can be recognized by a web browser. Finally boilerplate html tags are appended to the beginning and end of the list of html strings before it is written into a new html file.

There are many improvements that can made for this project. First of is to redesign how strings are parsed so that there can be nested classification of Markdown symbols on a line. As of now, each line can only be classified as one Markdown type because the parsing function is only search for the first special symbol that it can find that matches a Markdown symbol. Another improvement would be to use another regular expression library for Ocaml. As mentioned previously, the limitations of the Str library made it difficult to fully implement all the Markdown types for Common Mark and the Github Flavored Markdown specifications. Research into another library may be able to solve these issues.