Markdown Converter

Elijah Augustin

Why A Markdown Converter?

There are plenty of Markdown to HTML converters/parsers in other languages such as Python, Ruby, Javascript, etc.

However this project is still a challenging task and a good way to test your programming skills.

Background Knowledge

The concept of Parsers were first introduced in 1956 by Noam Chomsky in his paper **"Three Models for the Description of Language"**

The Chomsky Hierarchy for Formal Grammars

- Regular - Anything that can match with a regular expression
- Context-free- Can be represented as a PDA (maintain some state with a stack). Most programmning languages are context free.
- Context-senstive 0 Can be represented by an automata without using more memory than the length of the input
- Recursively enumerable - Anything that can solved by a computer (except the halting problem)

High-Level Process

- Read Lines from File into a List of Stings
- Convert each line into its proper Markdown data type/s
- Convert each Makrdown data type into its proper html string
- Write strings into a new file

Types

```
type markdownClassfication =
    | Heading1 of string
    | Heading2 of string
    | Heading3 of string
    | Heading4 of string
    | Heading5 of string
    | Heading6 of string
    | Paragraph of string
    | HorizontalLine of string
    | UnOrderedListItem of string
    | OrderedListItem of string
    | Code of string
    | BlockquoteItem of string
    | Blockquote of (string list)
    | UnOrderedList of (string list)
    | OrderedList of (string list)
    | Unknown of string
    | Empty
```

Ocaml String Regex Recipes

- Using the Str library
- Uses a limited Subset of Regular Expression
- Cant use some special symbols

```
let h1_recipe = Str.regexp "^#"
let h2_recipe = Str.regexp "^##"
let h3_recipe = Str.regexp "^###"
let h4_recipe = Str.regexp "^####"
let h5_recipe = Str.regexp "^#####"
let h6_recipe = Str.regexp "^######"

let unordered_list_recipe = Str.regexp "\\*\|\\-\|\\+"
let ordered_list_recipe = Str.regexp "[0-9]\\."

let code_recipe = Str.regexp "\\`"

let blockquote_recipe = Str.regexp "\\>"

let horizontal_line_recipe = Str.regexp
"\\(\\_\\_\\_\\)+\|\\(\\-\\-\\-\\)+\|\\(\\*\\*\\*\\)+"
```

Checking the first character of a line

The first character in a line can be interpreted to be many different Markdown data types

```ocaml
let map_tag line_string =
  if String.length line_string = 0 then
    Empty
  else
    let first_char = String.get line_string 0 in
    match first_char with
    | '#' -> check_heading_level line_string
    | '>' -> BlockquoteItem line_string
    | ' ' -> check_begining_whitespace line_string
    | _ -> check_other_options line_string
```

If you are converting each line, what About Markdown Data Types that span multiple line?

So in this implementation, we have three types that can span multiple lines (BlockquoteItem, UnOrderedListItem, and OrderedListItem)

When the converter first encounters one of these types, it checks the lines after in order to group them together into a larger type

```
let rec group_list_items classification_list =
  match classification_list with
  | [] -> []
  | (hd::tl) ->
      match hd with
      | BlockquoteItem x -> group_list_items
      (group_blockquote_items (Blockquote []) x tl)
      | UnOrderedListItem x -> group_list_items
      (group_unordered_items (UnOrderedList []) x tl)
      | OrderedListItem x -> group_list_items
      (group_ordered_items (OrderedList []) x tl)
      | _ ->  hd::group_list_items tl
  | _ -> classification_list
```

Example Of the UnOrderedList Grouping

```
let rec group_unordered_items unordered_item_object start_item
classification_list =
  match unordered_item_object with
  | UnOrderedList x1 ->
    match classification_list with
    | (hd::tl) ->
      match hd with
      | UnOrderedListItem x2 ->
        group_unordered_items (UnOrderedList (start_item::x1))
      | _ -> (UnOrderedList
      (start_item::x1))::hd::classification_list
    | _ -> classification_list
```

Issues I encountered

- As previously stated, Ocaml Str library uses a very limited Regular Expression syntax, so I had to get creative with how seached for sequences of strings
- Nested data types
- Checking for whitespaces

Lines of Code - 433 (as of May 4th)