Cameron Himes and Chase Faine

Dr. Andrew Polonsky

C S 3490

November 27, 2021

Pseudocode for Main

```
main :: IO ()

main = do

get system args

get file names

show input file name

show output file name

open the file

get the text from the file

print the input text

lex the input text

print the lexed data

parse the lexed data

print the parsed data

convert parsed data to html

write html code to file
```

Key Functions

getFiles

```
getFiles :: [String] -> (String, String)

getFiles [] = error "Error: No files provided." -- no files

getFiles [i] = error "Error: No output file provided." -- one file

getFiles [i, o] = (i, o) -- two files

getFiles (i : o : xs) = error "Error: Too many files provided." -- N files
```

This gets the names of the input and output files and will generate errors if anything is incorrect.

lexer

```
lexer :: String -> [Token]

lexer s = map classify (splitAtWords (preproc (convertSpacesToTabs s)))
```

This converts an entire file into a list of tokens. It has two helpers: `preproc` and `classify` which handle the classification of tokens. It also uses `splitAtWords` and `convertSpacesToTabs` to handle some extra processing.

parser

```
parser :: [Token] -> [Block]

parser input = sr input []
```

This is the parser. It converts tokens to blocks. It calls a single helper, `sr`, to generate the blocks.

sr

```
sr :: [Token] -> [Token] -> [Block]

sr (Err s : input) _ = error ("Lexical error: " ++ s) -- error case

sr [] [PB b] = [b] -- promote the last block element

-- several pattern matches for reduction

sr (i:input) stack = sr input (i:stack) -- shift stack

sr [p] stack        = error (show stack) -- ran out of options
```

This is the primary function of the parser that is the shift-reduce helper. This function which takes in a list of lexed tokens and a stack and converts the tokens into blocks which will be used to form a valid HTML structure.

structureToHTML

```
structureToHTML :: [Block] -> String

-- for each type of block, make an html element and recurse between the tags
```

This is the key function that takes in the correctly parsed markdown code and converts it into valid HTML code.

generateHTML

```
generateHTML :: [Block] -> String

-- generate the top and bottom of the html file and call structureToHTML between the b
```

This generates the HTML header and body code. It inserts the output from `generateHTML` inside the body tags to form a properly formatted HTML page.

Auxiliary Functions

isValidOrderedList

```
isValidOrderedList :: String -> Bool
```

This is a helper function that will check for ordered lists.

convertSpacesToTabs

```
convertSpacesToTabs :: String -> String
```

This is a helper function that will convert four spaces into one tab.

preproc

```
preproc :: String -> String
```

This is a helper function that will add spaces in between symbols so that tokens can be lexed correctly.

classify

```
classify :: String -> Token
```

This is a helper function for the lexer that will convert a single string to the correct token.

removeSpaceFront

```
removeSpaceFront :: String -> String
```

This is a helper function that removes the preceding space from a string (ex: " foo" -> "foo").

splitAtWords

```
splitAtWords :: String -> [String]
```

This is our own version of the Haskell `words` function. It will split a string into arrays that are delimited by spaces. The main difference is that control characters are not delimiters and will stay in the a returned string element.

splitAtWords'

```
splitAtWords' :: String -> [String]
```

This is a helper function for custom words function that does not remove tabs or newlines

splitAtBlocks

```
splitAtBlocks :: [Token] -> [[Token]]
```

This is a helper function wrapper for splitAtBlocks that removes empty blocks from the list before returning.

splitAtBlocks'

```
splitAtBlocks' :: [Token] -> [[Token]]
```

This is a helper function that splits a token list into several lists for each block element.