SimpleMD

A simple Markdown editor, written in Haskell

Xiangdong Zeng & Xuyang Duan

December 31, 2019 @ Fudan University

Introduction

What's Markdown?

Markdown is a **lightweight markup language** that you can use to add formatting elements to plaintext text documents.

Introduction

What's Markdown?

Markdown is a **lightweight markup language** that you can use to add formatting elements to plaintext text documents.

What's SimpleMD?

- A simple Markdown parser and editor
- Written in Haskell
- Based on web technology
- Support some extend features

Markdown syntax

```
<!-- Heading -->
# H1
## H2
### H3

<!-- Ordered list -->
1. First item
2. Second item
3. Third item

<!-- Unordered list -->
- First item
- Second item
  - Third item

<!-- Blockquote -->
> Blockquote
>
> - With items

<!-- Horizontal rule -->
___
```

```
<!-- Fenced code block -->
```c
int main() {
  printf("Hello, world!\n");
  return 0;
}
```

<!-- Inline text semantics -->
Bold, *italicized* and
~~deleted~~ text.

<!-- Link -->
[title](https://www.example.com)

<!-- Image -->
![alt text](image.jpg)

<!-- HTML blocks -->
<table>
  <tr><td>Hello</td></tr>
</table>
```

Extended features

- Super-/subscripts:
  - `19^th^ H~2~O`
  - 19th H_2O
- Highlight elements:
  - `==Highlight== and ++inserted++ text`
  - ==Highlight== and ++inserted++ text
- Math formulas (via KaTeX):
  - `$\sin^2\theta+\cos^2\theta=1$`
  - $\sin^2\theta+\cos^2\theta=1$
- Emoji:
  - `:smiley: :family_man_woman_girl_boy: :tada: :cn:`
  - 😃 👨‍👩‍👧 🎉 🇨🇳

Program structure

- Draw the initial page, via `Threepenny-GUI` library

Program structure

- Draw the initial page, via `Threepenny-GUI` library
- Get value of the input text

Program structure

- Draw the initial page, via `Threepenny-GUI` library
- Get value of the input text
- Parse input string into Markdown AST:
  - Split lines
  - Parse block elements: paragraph, code block, ordered/unordered list, blockquote, etc.
  - Parse inline elements: strong & emphasis, inline code, link, emoji, etc.

Program structure

- Draw the initial page, via `Threepenny-GUI` library
- Get value of the input text
- Parse input string into Markdown AST:
  - Split lines
  - Parse block elements: paragraph, code block, ordered/unordered list, blockquote, etc.
  - Parse inline elements: strong & emphasis, inline code, link, emoji, etc.
- Convert Markdown AST into HTML:
  - Add tags and attributes
  - Sanitize and add typographic replacements (quotation marks, dashes, etc.)

Program structure

- Draw the initial page, via `Threepenny-GUI` library
- Get value of the input text
- Parse input string into Markdown AST:
  - Split lines
  - Parse block elements: paragraph, code block, ordered/unordered list, blockquote, etc.
  - Parse inline elements: strong & emphasis, inline code, link, emoji, etc.
- Convert Markdown AST into HTML:
  - Add tags and attributes
  - Sanitize and add typographic replacements (quotation marks, dashes, etc.)
- Put the HTML string on web page with a pretty style sheet
  - Load JavaScript code for Math rendering & syntax highlight

File structure

```
./
├── simplemd.cabal
├── Setup.hs
├── src/
│   Main.hs                    -- Entry point of program
│   └── Gui.hs                  -- Draw the GUI, via Threepenny-GUI library
│       ├── Svg.hs              -- Draw SVG icons
│       └── Html.hs             -- Transform Markdown AST into HTML
│           └── Parse.hs        -- Markdown parser (core of the program)
│               └── Emoji.hs    -- Emoji database
│
└── static/
    index.html                 -- Web page for the editor
    simplemd.css               -- Style sheet
```

Markdown AST

```haskell
newtype Markdown = Markdown { unMarkdown :: [BlockElem] }
  deriving (Eq)

data BlockElem =
    Hrule
  | Heading { level  :: Int,  elems  :: [InlineElem] }
  | Para    { isOpen :: Bool, elems  :: [InlineElem] }
  | Pre     { isOpen :: Bool, lang   :: String, text :: String }
  | Ulist   { isOpen :: Bool, items  :: [ListItem] }
  | Olist   { isOpen :: Bool, items  :: [ListItem], start :: Int }
  | Quote   { isOpen :: Bool, elems' :: [BlockElem] }  -- Recursive
  deriving (Eq)

data InlineElem =
    Plain    { content :: String }
  | Code     { content :: String }
  | Em       { content :: String }
  | Link     { content :: String, url :: String }
  ...
  deriving (Eq, Show)
```

Parser (monadic & recursive)

```
parse :: [String] -> Markdown
parse = foldl parseMarkdown $ Markdown []

parseMarkdown :: Markdown -> String -> Markdown
parseMarkdown (Markdown []) s = Markdown [result]
  where Left result = parseHeading s
                  >>= parseHrule >>= ...
parseMarkdown (Markdown mdElements) s =
  case last mdElements of
    Pre { isOpen = True, .. } ->
      Markdown $ init mdElements ++ nextPre lang text s
    ...
    _ -> Markdown $ if null s
      then mdElements
      else mdElements ++ (unMarkdown $ parseMarkdown (Markdown []) s)

parseInline :: String -> [InlineElem]
parseInline "" = []
parseInline s = result
  where Left result = parseCode s
                  >>= parseEmoji >>= ...
```

Regular expression

```haskell
{-# LANGUAGE QuasiQuotes #-}
import qualified Text.Regex as Regex
import Text.RawString.QQ

parseHeading :: String -> Either BlockElem String
parseHeading = mkParser headingPatt parseHeading_

mkParser :: Regex.Regex -> ([String] -> a)
                        -> (String -> Either a String)
mkParser pattern parser = \s -> case Regex.matchRegex pattern s of
  Just x  -> Left $ parser x
  Nothing -> Right s

headingPatt :: Regex.Regex
headingPatt = Regex.mkRegex [r|^(#{1,6}) (.*)|]  -- (#...#), (content)
```

Emoji

```haskell
-- Parse.hs
parseEmoji_ :: [String] -> [InlineElem]
parseEmoji_ x = parseInline_ x $ Plain { content = emoji }
  where name  = x !! 1
        emoji = case Map.lookup name emojiMap of
          Just y  -> y
          Nothing -> ":" ++ name ++ ":"
```

```haskell
-- Emoji.hs
import qualified Data.Map as Map

emojiMap :: Map.Map String String
emojiMap = Map.fromList [ ("-1",   "&#x1f44e;")
                        , ("+1",   "&#x1f44d;")
                        , ("100",  "&#x1f4af;")
                        , ("1234", "&#x1f522;")
                        , ...
                        , ("zzz",  "&#x1f4a4;")
                        ]  -- Data from https://api.github.com/emojis
```

GUI

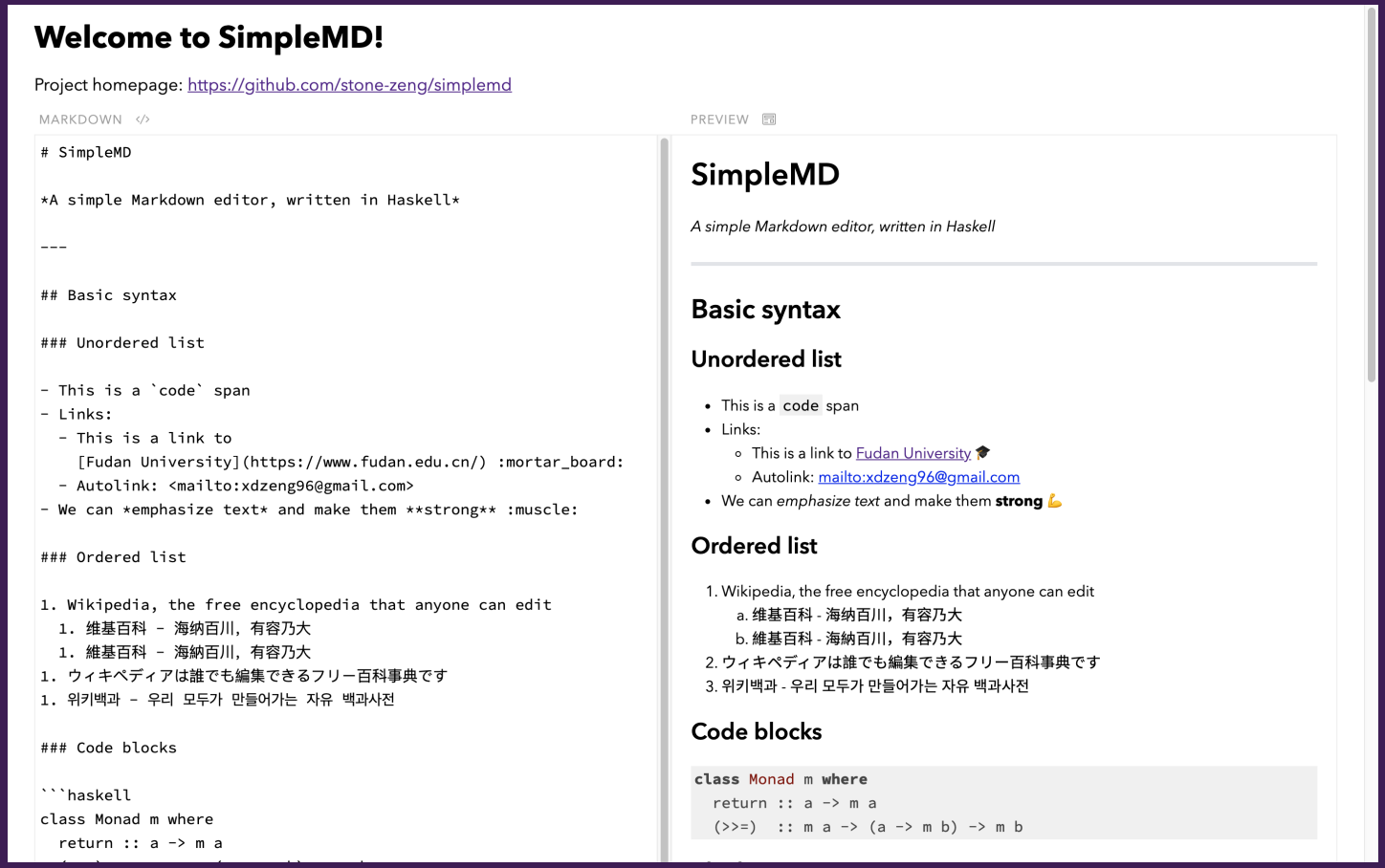
```haskell
import Graphics.UI.Threepenny ((#), (#.), (#+))
import qualified Graphics.UI.Threepenny as UI

gui :: UI.Window -> UI.UI ()
gui window = void $ do
  return window # UI.set UI.title "SimpleMD"

  title <- UI.h1
    # UI.set UI.text "Welcome to SimpleMD!"

  -- More elements

  UI.on UI.valueChange mdInput $ \_ -> do
    markdownText <- UI.get UI.value mdInput
    UI.element mdOutput
      # UI.set html (markdownToHtml markdownText)
    UI.runFunction $ UI.ffi jsCode  -- Load some JavaScript code

html :: UI.Attr UI.Element String
html = UI.mkReadWriteAttr get set
  where get   el = UI.callFunction $ UI.ffi "$(%1).html()" el
        set v el = UI.runFunction  $ UI.ffi "$(%1).html(%2)" el v
```

HTML & CSS

```html
<!doctype html>
<head>
  <meta charset="UTF-8">
  <link rel="stylesheet" type="text/css" href="static/simplemd.css" />
  <script src="haskell.js"></script>
  <!-- More CSS and JavaScript code -->
  <script type="text/javascript" charset="utf-8">
    Haskell.initFFI();
  </script>
</head>
<body></body>
</html>
```

```css
body {
  font: 400 16px/1.6 'Avenir Next', 'Source Han Sans SC', sans-serif;
  font-feature-settings: 'kern';
  font-kerning: normal;
  word-wrap: break-word;
}
...
```

Demo


Welcome to SimpleMD!
Project homepage: https://github.com/stone-zeng/simplemd
MARKDOWN
SimpleMD
A simple Markdown editor, written in Haskell

Basic syntax
Unordered list
- This is a `code` span
- Links:
  - This is a link to
    [Fudan University](https://www.fudan.edu.cn/) :mortar_board:
  - Autolink: <mailto:xdzeng96@gmail.com>
- We can *emphasize text* and make them **strong** :muscle:
Ordered list
1. Wikipedia, the free encyclopedia that anyone can edit
  1. 维基百科 - 海纳百川，有容乃大
  1. 維基百科 - 海納百川，有容乃大
1. ウィキペディアは誰でも編集できるフリー百科事典です
1. 위키백과 - 우리 모두가 만들어가는 자유 백과사전
Code blocks
```haskell
class Monad m where
  return :: a -> m a
PREVIEW
SimpleMD
A simple Markdown editor, written in Haskell
Basic syntax
Unordered list
This is a code span
Links:
This is a link to Fudan University
Autolink: mailto:xdzeng96@gmail.com
We can emphasize text and make them strong
Ordered list
1. Wikipedia, the free encyclopedia that anyone can edit
a. 维基百科 - 海纳百川，有容乃大
b. 維基百科 - 海納百川，有容乃大
2. ウィキペディアは誰でも編集できるフリー百科事典です
3. 위키백과 - 우리 모두가 만들어가는 자유 백과사전
Code blocks
class Monad m where
return :: a -> m a
(>>=) :: m a -> (a -> m b) -> m b
```

Known issues & TODO

- ☹ Not fully supprt CommonMark Spec:
  - Only support 2-level `<ul>`/`<ol>`
  - Only `<ul>`/`<ol>` are allowed inside `<ul>`/`<ol>`
  - Rules for `<em>`, `<strong>`, `<a>`, etc. are not complete
- ☹ Syntax for extended features may clash with normal text:
  - E.g. `:` for emoji, `$` for Math formulas, `<`/`>` for HTML tags
- ☹ Some special characters may break regex matching
- ☹ Not support other HTML event (e.g. `onload`, `onpaste`)

Known issues & TODO

- ☹ Not fully supprt CommonMark Spec:
  - Only support 2-level `<ul>`/`<ol>`
  - Only `<ul>`/`<ol>` are allowed inside `<ul>`/`<ol>`
  - Rules for `<em>`, `<strong>`, `<a>`, etc. are not complete
- ☹ Syntax for extended features may clash with normal text:
  - E.g. `:` for emoji, `$` for Math formulas, `<`/`>` for HTML tags
- ☹ Some special characters may break regex matching
- ☹ Not support other HTML event (e.g. `onload`, `onpaste`)

- ☺ Support more rules:
  - Indent code block, more generic nested list
  - Auto-detected URL
  - Tables, footnotes, toc, etc.
- ☺ Real-time highlight for input
- ☺ Support conversion to LaTeX, reStructuredText, etc.

References

- John Gruber. Markdown
  - https://daringfireball.net/projects/markdown
- CommonMark Spec, version 0.29
  - https://spec.commonmark.org/0.29
- GitHub. Basic writing and formatting syntax
  - https://help.github.com/en/github/writing-on-github/basic-writing-and-formatting-syntax
- Dillinger - Online Markdown editor
  - https://dillinger.io
- `markdown-it` - Markdown parser, done right
  - https://github.com/markdown-it/markdown-it
- `threepenny-gui` - GUI framework that uses the web browser as a display
  - https://hackage.haskell.org/package/threepenny-gui
- `regex-compat-tdfa` - Unicode support version of `Text.Regex`, using `regex-tdfa`
  - https://hackage.haskell.org/package/regex-compat-tdfa
- KaTeX - The fastest math typesetting library for the web
  - https://katex.org
- `highlight.js` - Javascript syntax highlighter
  - https://highlightjs.org

Thank you!

 stone-zeng/simplemd