The Mabel Literate Programming Tool

A Simple Tangler

M-CS-ME

Introduction

I wanted to take notes with executable code snippets inside, and I didn't want it to be glued to my text editor. This little project is supposed to (eventually) do that.

I chose markdown because it's simple, light weight, and has many implementations. A markdown engine or `pandoc` will be better than any weaver that I could write.

The source in `src/mabel.go` is generated with `mabel` through `mabel mabel.md > src/mabel.go`.

Imports and packages

I let the package name be `main` for now. For this file we will need the `os`, `bufio`, `strconv`, and `fmt` packages from the stdlib.

```go
package main

import (
    "os"
    "bufio"
    "fmt"
    "strconv"
)
```

Dealing with errors

Since we'll be doing a lot of file i/o, lets make a `check` function that will panic if it recieves an error.

```
func check(e error) {
    if e != nil {
        panic(e)
    }
}
```

Tangling Specific Source Blocks

The whole reason I started this project was so I could run code snippets from inside a text file. So I need to be able to tangle a specific source block.

Finding Source Blocks

Because I want to be able to print a specific source block, I first need to find where those source blocks are. So I want this function to return two arrays specifying the lines where source blocks begin `bg []int` and where they end `en []int`. Basically if a source block just opened, append the line number to `bg` and if it just closed append the line number to `en`.

```
func srcblks(filename string) (bg, en []int) {
    file, err := os.Open(filename)
    check(err)
    in := bufio.NewScanner(file)
    var open bool = false
    for i := 0; in.Scan(); i++ {
        ln := in.Text()
        if len(ln) >= 3 {
            if ln[:3] == "```" {
                open = !open
                if open {
                    bg = append(bg, i)
                } else {
                    en = append(en, i)
                }
            }
```

```
        }
    }
    return
}
```

Writing Parts of A Buffer to Stdout

Write the `bg`th to `en`th elements of a buffer to stdout

```
func write(buf []string, bg, en int) {
    for _, i := range buf[bg+1:en] {
        fmt.Println(i)
    }
}
```

Tangling Specific and General Ones

Basically, if `blk` is `-1` tangle the entire buffer, if not tangle the `blk`th source block.

```
func tangle(in string, blk int) {
    file, err := os.Open(in)
    check(err)
    f := bufio.NewScanner(file)
    bg, en := srcblks(in)
    var buf []string
    for f.Scan() {
        buf = append(buf, f.Text())
    }
    if blk == -1 {
        for i := 0; i < len(bg); i++ {
            write(buf, bg[i], en[i])
        }
    } else {
        write(buf, bg[blk], en[blk])
    }
}
```

Main and dealing with input

Implement an user interface using command line args for the `tangle` function.

```
func main() {
    if len(os.Args) > 2 {
        blk, _ := strconv.Atoi(os.Args[2])
        tangle(os.Args[1], blk)
    } else if len(os.Args) > 1 {
        tangle(os.Args[1], -1)
    } else {
        fmt.Println("error: no input files")
    }
}
```

What's next?

- ☒ Ability to print to stdout a selected group of code blocks (like `org-babel`).
- ☐ Add concurency for speed.
- ☐ Add a way to compile/run a specific code block and write the output in a file.
- ☐ Add a way to configure specific compilers/interpreters through a text file.