The clojure-pamphlet package*

Ernesto Lanchares Sanchez
e.lancha98@gmail.com

September 9, 2019

Abstract

A package to make beautiful literate programming documents. The system is based on clojure's pamphelt files.

1 Introduction

Literate Programming is a programming paradigm that changes the goal of the program:

> Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.

Since it's creation, many systems have been created to accomplish this goal, systems like `WEB`, `CWEB`, `noweb`, ... This systems all share the same structure: There is one source file and two programs needed, one that extracts the code to be compiled and other that extracts the literature (usually a TEX/LATEXfile) to be compiled.

This is the reason that we preffer clojure's system where there is one LATEXsource file that can be compiled and one program to extract the code to be compiled. Although the clojure system is fine as it is, we think the resulting documentation can be a bit prettier hence this package. The difference in using this package or the clojure system as a developer is non-existent, however we think the documentation with our system is a bit more readable. It also automaticly adds hyperlinks to the output in order to make it easier to navigate when viewing in a computer.

2 Usage

The `clojure-pamphlet` system is designed to be as simple as possible, so this package only provides one environment and one command. The system is based

*This document corresponds to clojure-pamphlet v1.o, dated 2019/07/08.

arround code chunks. This chunks are the part of the documents that contain code and can be extracted. The chunks also contain a name so that they can be referenced in the document and in other chunk blocks. The name is also needed for the tangler to extract the chunk.

2.1 `chunk` environment

`chunk` The chunk environment is used to define code blocks. This code blocks have a name and are what the tangler will actually output. It is based on the listings package, so all style formats that yo can apply to `lstlisting` environments, you can also apply to chunk environments by simply using the command `lstset`.

Here's an example of a code usage and it's output

```
\begin{chunk}{main-routine}
int main(int argc, char* argv[]) {
  printf("Hello World.\n");
}
\end{chunk}
```

— **main-routine** —

```
int main(int argc, char* argv[]) {
  printf("Hello World.\n");
}
```

——

`\getchunk` Also, inside the `chunk` environment, you can use the `\getchunk` command, which includes the referenced chunk at that exact same spot. The `\getchunk` command requires to be in it's own separate line. Let's illustrate this with an example:

```
\begin{chunk}{onechunk}
CHUNK ONE
\end{chunk}
\begin{chunk}{otherchunk}
PREVIOUS
\getchunk{onechunk}
POST
\end{chunk}
```

— **print** —

```
printf("Hello World");
```

—*Used by, main*—

— **main** —

```
int main(int argc, char* argv[]) {
    ⟨print⟩
}
```

——

2.2 The tangler

The tangler is designed to be as simple to use as possible. You just need to provide it with a LaTeX`clojure-pamphlet` file and a code chunk to extract. Then the tangler will output the code to be compiled in the standard output. For example in order to extract the main chunk of code we will need to run

```
pamphletangler [filename.tex] main
```

And the output will be

```
int main(int argc, char* argv[]) {
      printf("Hello World");
}
```