Knit

by Todd Coram (todd@maplefish.com)

April 30, 2008

Contents

Version 1.3a

---

1 Introduction

Knit converts awk, C/C++ (and maybe Tcl, Java and other languages that use `/* */` or `#` style comments) source into AFT http://www.maplefish.com/todd/aft.html documents. You are currently looking at the AFT processed source code to Knit. Knit is written in itself and Gawk. You can downloaded the most up to date source here http://www.maplefish.com/todd/knit.awk. A PDF version of this file resides here http://www.maplefish.com/todd/knit.pdf.

Knit is in the spirit of Literate Programming http://www.literateprogramming.com, but is not foolish enough to claim that it is an example of such.

Both *C style* comments (`/* */`) and single line awk/Tcl/etc comments (`#`) are used to indicate AFT text. For any language using these commenting conventions, Knit fits neatly into the comments of source while keeping the source syntatically legit.

Knit is a very, very poor man's Literate Programming tool.

Below, you can peruse the actual Gawk source code for Knit. This document is the result of running:

```
$ gawk -f knit.awk knit.awk >knit.aft && aft knit.aft
```

2 Code

2.1 BEGIN

First, we parse the only (optional) argument we currently support: `-p`. The argument `-p` signifies that we will attempt to *pretty print* the code. In AFT, this is faciliated by `^<<Filter` which signifies that certain formatting is interpreted in the otherwise verbatim text. In particular, the listed `c_keywords` are bolded.

Since LaTeX is funny about non-escaped braces in what we call filtered verbatim, we set up symbols for left and right brace to be used in place of every actual brace found in the code.

Also, we use two control variables for our parser: `text` and `in_block`. These are used to track whether we are in *text* mode or *code* mode respectively.

```
BEGIN {
  if (ARGV[1] ~ /-p/) {
    pretty = 1;
    ARGV[1]="";

    print "#---SET-LaTeX lb=\\{";
    print "#---SET-HTML  lb={";
    print "#---SET-LaTeX rb=\\}";
    print "#---SET-HTML  rb=}";
  }
  text = 0; in_block = 0;

  start_block = pretty ? "^<<Filter" : "^<<";
  c_keywords = "#ifdef|#if|#define|int|char|unsigned|long|void|while|do" \
               "|if|else|#include|typedef|const";
}
```

2.2 Awk Comments

Awk is particularly easy to handle as it only supports single line comments. So, just about all of the text matching is done here. If the file is awk source, then we look for a leading `#` to signal that we should get out of code *block* mode and treat the line as text.

We strip out the `#` character and spit out the rest of the line.

```
FILENAME ~ /\.awk$/ && /^[ \t]*#/ {
  if (in_block) printf("^>>\n"); in_block = 0;
  gsub(/^[ \t]*#[ ]?/,"");
  print $0; next;
}
```

2.3 Begin C Comment

This pattern handles the C style beginning comment (/*). When encountered we force *text* mode and leave *block* mode (if we are in it). This *text* mode is only entered when a comment begins a line.

As an added bonus, if the start of comment is immediate followed by text, then 1 word of that text is used as a third level section header.

For example:

```
/* foobar-handler
  This is the foobar-handler function... blah, blah...
*/
```

This is a convenient way of putting function names into the AFT table of contents. _Note: A current bug in AFT results in underscore (_) not being handled properly in section names. They are currently converted here into dashes (-).

```
/^[ \t]*\/\*/ {
  if (in_block) printf("^>>\n"); text = 1; in_block = 0;
  if ($NF > 1) {
    gsub(/_/,"-",$2);
    printf("***%s\n", $2);
  }
  next;
}
```

2.4 End C Comment

Look for end of C style comment (*/) and take us out of text mode.

```
/^[ \t]*\*\// {
  text = 0; next;
}
```

2.5 C Comment Text

Handle AFT text mode. We are in a *comment*. To allow for the stylistic convention of preceding comments with bare stars (*), we strip them out. What this means for AFT style Section headers is reflected in the following example:

```
/*
 * *Section 1
 *  This is the body of
 * Section 1.
 even without a leading star
 */
```

We do this only when currently in text mode (previous comment start seen).

```
text {
  gsub(/[ ]\*[ ]*$/, "");
  gsub(/^ /,"");
  print $0; next;
}
```

2.6 C Code

If not in text mode, currently not in a block and not a blank line, then you may start a *verbatim* block.

```
!text && !in_block && $0 !~ /^[ ]*$/ {
  print start_block;
  in_block = 1;
}
```

If we are in a block, print out the line of text. If we are in *pretty* mode, then fix occurences of `_`, `|` and `~`, so that they are not interpreted as font face tricks (bold, teletype and small, respectively). Italicize C keywords and make C++ (`//`) and C style comments small.

Since braces are known to screw up TEX in filtered verbatim mode, we use previously embedded macros `lb` and `rb`.

```
in_block {
  if (pretty) {
   gsub(/\_/,"\\_");
   gsub(/\|/,"\\|");
   gsub(c_keywords, "''&''");
   gsub(/\/\/.*/, "~&~");
   gsub(/\/\*.+\*\//, "~&~");
   gsub(/\{/,"%lb%");
   gsub(/\}/,"%rb%");
  }
   print $0; next;
}
```

3 Conclusion

So, here we have Knit. The awk support was really only put in so that Knit could be written in itself.

This system is a far cry from Knuth's Literate Programming, but it gives a taste of that discipline without requiring preprocessing of soure code. Knit is embedded in plain old comments.

Enjoy!