

Sample Literate Programming

BY SAM LIDDICOTT

Using $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ and Newfangle

Abstract

Literate Programming will not be explained here but it will be shown as a way narrate program development showing the progression of ideas as is natural for humans to understand, leaving the generation of programs from those ideas to the machine.

$\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ is used as the editor and NEWFANGLE^1 as the extraction tool.

The stylesheet is demonstrated here, but NEWFANGLE has not yet been adapted to cope with $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ files.

Here we will explain the popular *hello world* written in C.

Message

Here is the message that we wish to give to world:

```
1a <message[1], lang=txt>≡  
    Hello World!
```

Because we did not press enter in the literate programming environment, this is not a line but a line-fragment and has no line number displayed.

Note 1. The header contains the chunk name *message* as well as the language of the listing. The beginning of the header gives the id to this *chunk* of code — 1a — which suggests to the reader that it is the first chunk on page 1.

Displaying the message

We have a few choices available. The message can be output with `printf`, like this:

```
1b <message-printf[1], lang=cpp>≡  
1 printf("«message»\n");
```

or even with `puts`:

```
1c <message-puts[1], lang=cpp>≡  
1 puts("«message»\n");
```

But, we prefer `printf` which is more traditional.

Note 2. These last two chunks have a different letter for the chunk reference even though they may be on the same page.

Note 3. They also contain a line number, because they are intended to be a full line of text.

The main function

We have to enclose this line in the standard C *main* function, like this:

```
1d <main[1], lang=cpp>≡  
1 int main(int argc, char** argv) {  
2     «message-printf»  
3     return 0;  
4 }
```

1. <http://new.fangled.org>

Note the `printf` statement defined in 1b has been included in 1d line 2.

The statement `return 0;` lets the operating system know that the function completed successfully — which is a bit of a presumption as we don't check if `printf` is successful.

Note 4. The line numbering starts again for each named chunk.

Header files

On my system, both `printf` and `puts` both require the header `stdio.h`; so this line becomes the first line of our file.

```
2a <hello-world.c[1], lang=cpp> ≡ 2b ▷  
1 | #include stdout.h
```

Note 5. The right of the header contains a link to chunk 2b which is the next chunklet of this named chunk.

The include statement is followed by our main function that we defined in 1d.

```
2b <hello-world.c[2] ↑2a, lang=cpp> + ≡ <2a 2c ▷  
2 | «main»
```

Note 6. This time the header also contains a link to chunk 2a which is the previous chunklet of this named chunk.

And a final good-bye comment.

```
2c <hello-world.c[3] ↑2a, lang=cpp> + ≡ <2b  
3 | /* thats the end, folks */
```

Note 7. Like the first chunks we looked at, there are no further chunklets with the same name and so there is no link shown in the header, although there is a link to the previous chunklet.

Updates

Because of the amount of page-referencing going on it can sometimes require the document to be updated three (or maybe more) times to get all of the links and references right.

First a chunk has to discover it is on a new page.

The second time all the chunks can be given the correct id.

The third time, all references to the correct id can be updated.

A forth and additional times may be required if the page-breaking was changed as a result of a change in label size — although this is only likely to occur if a reference occurred in the text of the document for the references in the header have enough space so as not to affect layout.