LuaTEX says goodbye to Pascal

Abstract
LuaTEX 0.50 features a complete departure from Pascal source code. This article explains a little of the why and how of this change.

Introduction

For more than a quarter of a century, all implementations of the TEX programming language have been based on WEB, a literate programming environment invented by Donald Knuth. WEB input files consist of a combination of Pascal source code and TEX explanatory texts, and have to be processed by special tools to create either input that is suitable for a Pascal compiler to create an executable (this is achieved via a program called `tangle`) or input for TEX82 to create a typeset representation of the implemented program (via a program called `weave`).

A WEB file is split into numerous small building blocks called 'modules' that consist of an explanatory text followed by source code doing the implementation. The explanations and source are combined in the WEB input in a way that attempts to maximise the reader's understanding of the program when the typeset result is read sequentially. This article will, however, focus on the program source code.

Pascal WEB

The listing that follows shows a small part of the WEB input of TEX82, defining two 'modules'. It implements the function that scans for a left brace in the TEX input stream, for example at the start of a token list.

```
@ The |scan_left_brace| routine is called when a left brace is supposed to
be the next non-blank token. (The term ``left brace'' means, more precisely,
a character whose catcode is |left_brace|.) \TeX\ allows \.{\\relax} to
appear before the |left_brace|.

@p procedure scan_left_brace; {reads a mandatory |left_brace|}
begin @<Get the next non-blank non-relax non-call token@>;
if cur_cmd<>left_brace then
  begin print_err("Missing { inserted");
@.Missing \{ inserted@>
  help4("A left brace was mandatory here, so I've put one in.")@/
    ("You might want to delete and/or insert some corrections")@/
    ("so that I will find a matching right brace soon.")@/
    ("(If you're confused by all this, try typing `I}' now.)");
  back_error; cur_tok:=left_brace_token+"{"; cur_cmd:=left_brace;
  cur_chr:="{"; incr(align_state);
  end;
end;

@ @<Get the next non-blank non-relax non-call token@>=
repeat get_x_token;
until (cur_cmd<>spacer)and(cur_cmd<>relax)
```

It would take too much space here to explain everything about the WEB programming, but it is necessary to explain a few things. Any @ sign followed by a single character introduces a WEB command that is interpreted by tangle, weave or both. The commands used in the listing are:

@ This indicates the start of a new module, and the explanatory text that follows uses special TEX macros but is otherwise standard TEX input (this command is followed by a space).
@p This command starts the Pascal implementation code that will be transformed into the compiler input.
@< When this command is seen after a @p has already been seen, the text up to the following @> is a reference to a named module that is to be inserted in the compiler output at this spot. If a @p has not been seen yet, then instead it defines or extends that named module.
@. This defines an index entry up to the following @>, used by weave and filtered out by tangle.
@/ This is a command to control the line breaks in the typeset result generated by weave.

If you are familiar with Pascal, the code above may look a bit odd to you. The main reason for that apparent weirdness of the Pascal code is that the WEB system has a macro processor built in. The symbol help4 is actually one of those macros, and it handles the four sets of parenthesized strings that follow. In expanded form, its output would look like this:

```
begin
  help_ptr:=4;
  help_line[3]:="A left brace was mandatory here, so I've put one in.";
  help_line[2]:="You might want to delete and/or insert some corrections";
  help_line[1]:="so that I will find a matching right brace soon.";
  help_line[0]:="(If you're confused by all this, try typing 'I}' now.)";
end
```

The symbol print_err is another macro, and it expand into this:

```
begin if interaction=error_stop_mode then wake_up_terminal;
  print_nl("! "); print("Missing { inserted");
  end
```

Tangle output

Now let's have a look at the generated Pascal.

```
{:381}{403:}procedure scanleftbrace;begin{404:}repeat getxtoken;
until(curcmd<>10)and(curcmd<>0){:404};
if curcmd<>1 then begin begin if interaction=3 then;
if filelineerrorstylep then printfileline else printnl(262);print(671);
end;begin helpptr:=4;helpline[3]:=672;helpline[2]:=673;helpline[1]:=674;
helpline[0]:=675;end;backerror;curtok:=379;curcmd:=1;curchr:=123;
incr(alignstate);end;end;{:403}{405:}procedure scanoptionalequals;
```

That looks even weirder! Don't panic. Let go through the changes one by one.

- First, tangle does not care about indentation. The result is supposedly only read by the Pascal compiler, so everything is glued together.
- Second, tangle has added Pascal comments before and after each module that give the sequence number of that module in the WEB source: those are 403 and 404.
- Third, tangle removes the underscores from identifiers, so scan_left_brace became scanleftbrace etcetera.

- Fourth, many of the identifiers you thought were present in the WEB source are really macros that expand to constant numbers, which explains the disappearance of `left_brace`, `spacer`, `relax`, `left_brace_token`, and `error_stop_mode`.
- Fifth, macro expansion has removed `print_err`, `help4` and `wake_up_terminal` (which expands to nothing).
- Sixth, WEB does not make use of Pascal strings. Instead, the strings are collected by `tangle` and output to a separate file that is read by the generated executable at runtime and then stored in a variable that can be indexed as an array. This explains the disappearance of all the program text surrounded by `"`.
- Seventh, addition of two constants is optimized away in some cases. Since single character strings in the input have a fixed string index (its ASCII value), `left_brace_token+"{"` becomes `379` instead of `256+123`.

Web2c

Assuming you have generated the Pascal code above via `tangle`, you now run into a small practical problem: the limited form of Pascal that is used by the WEB program is not actually understood by any of the current Pascal compilers and has not for a quite some time. The build process of modern TEX distributions therefore make use of a different program called `web2c` that converts the `tangle` output into C code.

The result of running `web2c` on the above snippet is in the next listing.

```
void scanleftbrace ( void )
{
  do {
      getxtoken () ;
  } while ( ! ( ( curcmd != 10 ) && ( curcmd != 0 ) ) ) ;
  if ( curcmd != 1 )
  {
    {
      if ( interaction == 3 )
      ;
      printnl ( 262 ) ;
      print ( 671 ) ;
    }
    {
      helpptr = 4 ;
      helpline [3 ]= 672 ;
      helpline [2 ]= 673 ;
      helpline [1 ]= 674 ;
      helpline [0 ]= 675 ;
    }
    backerror () ;
    curtok = 379 ;
    curcmd = 1 ;
    curchr = 123 ;
    incr ( alignstate ) ;
  }
}
```

This output is easier to read for us humans because of the added indentation and the addition of parentheses after procedure calls, but does not significantly alter the code.

Actually, some more tools are used by the build process of modern TEX distributions, for example there is a small program that splits the enormous source file into a few smaller C source files, and another program converts Pascal `write` procedure calls to C `printf` function calls.

Issues with WEB development

Looking at the previous paragraphs, there are number of issues when using WEB for continuous development.

- Compiling WEB source, especially on a modern platform without a suitable Pascal compiler, is a fairly complicated and lengthy process requiring a list of dedicated tools that have to be run in the correct order.
- Literate programming environments like WEB never really caught on, so it is hard to find programmers that are familiar with the concepts, and there are nearly no editors that support its use with the editor features that modern programmers have come to depend on, like syntax highlighting.
- Only a subset of an old form of Pascal is used in the Knuthian sources, and as this subset is about the extent of Pascal that is understood by the `web2c` program, that is all that can be used.

  This makes interfacing with external programming libraries hard, often requiring extra C source files just to glue the bits together.
- The ubiquitous use of WEB macros makes the external interface even harder, as these macros do not survive into the generated C source.
- Quite a lot of useful debugging information is irretrievably lost in the `tangle` stage. Of course, TEX82 is so bug-free that this hardly matters, but when one is writing new extensions to TEX, as is the case in LuaTEX, this quickly becomes problematic.
- Finally, to many current programmers WEB source simply feels over-documented and even more important is that the general impression is that of a finished book: sometimes it seems like WEB actively discourages development. This is a subjective point, but nevertheless a quite important one.

Our solution

In the winter of 2008–2009, we invested a lot of time in hand-converting the entire LuaTEX code base into a set of C source files that are much closer to current programming practices. The big WEB file has been split into about five dozen pairs of C source files and include headers.

During conversion, quite a bit of effort went into making the source behave more like a good C program should: most of the WEB macros with arguments have been converted into C `#defines`, most of the numerical WEB macros are now C enumerations, and many of the WEB global variables are now function arguments or static variables. Nevertheless, this part of the conversion process is nowhere near complete yet.

The new implementation of `scan_left_brace` in LuaTEX looks like this:

```
/*
The |scan_left_brace| routine is called when a left brace is supposed
to be the next non-blank token. (The term ``left brace'' means, more
precisely, a character whose catcode is |left_brace|.) \TeX\ allows
\.{\\relax} to appear before the |left_brace|.
*/

void scan_left_brace(void)
{                               /* reads a mandatory |left_brace| */
    /* Get the next non-blank non-relax non-call token */
    do {
        get_x_token();
    } while ((cur_cmd == spacer_cmd) || (cur_cmd == relax_cmd));

    if (cur_cmd != left_brace_cmd) {
        print_err("Missing { inserted");
```

```
        help4("A left brace was mandatory here, so I've put one in.",
              "You might want to delete and/or insert some corrections",
              "so that I will find a matching right brace soon.",
              "If you're confused by all this, try typing 'I}' now.");
        back_error();
        cur_tok = left_brace_token + '{';
        cur_cmd = left_brace_cmd;
        cur_chr = '{';
        incr(align_state);
    }
}
```

I could write down all of the differences with the previously shown C code, but that is not a lot of fun so I will leave it to the reader to browse back a few pages and spot all the changes. The only thing I want to make a note of is that we have kept all of the WEB explanatory text in C comments, and we are actively thinking about a way to reinstate the ability to create a beautifully typeset source book.

Taco Hoekwater
taco (at) luatex dot org