The DVItype processor

(Version 3.6, December 1995)
1* Introduction. The DVItype utility program reads binary device-independent (“DVI”) files that are produced by document compilers such as TeX, and converts them into symbolic form. This program has two chief purposes: (1) It can be used to determine whether a DVI file is valid or invalid, when diagnosing compiler errors; and (2) it serves as an example of a program that reads DVI files correctly, for system programmers who are developing DVI-related software.

Goal number (2) needs perhaps a bit more explanation. Programs for typesetting need to be especially careful about how they do arithmetic; if rounding errors accumulate, margins won’t be straight, vertical rules won’t line up, and so on. But if rounding is done everywhere, even in the midst of words, there will be uneven spacing between the letters, and that looks bad. Human eyes notice differences of a thousandth of an inch in the positioning of lines that are close together; on low resolution devices, where rounding produces effects four times as great as this, the problem is especially critical. Experience has shown that unusual care is needed even on high-resolution equipment; for example, a mistake in the sixth significant hexadecimal place of a constant once led to a difficult-to-find bug in some software for the Alphatype CRS, which has a resolution of 5333 pixels per inch (make that 5333.33333333 pixels per inch). The document compilers that generate DVI files make certain assumptions about the arithmetic that will be used by DVI-reading software, and if these assumptions are violated the results will be of inferior quality. Therefore the present program is intended as a guide to proper procedure in the critical places where a bit of subtlety is involved.

The first DVItype program was designed by David Fuchs in 1979, and it went through several versions on different computers as the format of DVI files was evolving to its present form. Peter Breitenlohner helped with the latest revisions.

The banner string defined here should be changed whenever DVItype gets modified.

```pascal
define my_name ≡ 'dvitype'
define banner ≡ 'This is DVItype, Version 3.6' { printed when the program starts }
```

3* The binary input comes from dvi_file, and the symbolic output is written on Pascal’s standard output file. The term print is used instead of write when this program writes on output, so that all such output could easily be redirected if desired.

```pascal
define print(#) ≡ write(stdout, #)
define print_ln(#) ≡ write_ln(stdout, #)
```

program DVI_type(dvi_file, output);

```pascal
label { Labels in the outer block 4*}  
const { Constants in the outer block 5*}  
type { Types in the outer block 8*}  
var { Globals in the outer block 10}  
{ Define parse_arguments 112*}  
procedure initialize; { this procedure gets things started properly }  
  var i: integer; { loop index for initializations }  
  begin  
    kpse_set_program_name(argv[0], my_name); parse_arguments; print(banner);  
    print_ln(version_string); { Set initial values 11 }  
  end;
```

4* Label done is used when stopping normally.

```pascal
define done = 30 { go here when finished with a subtask }
```

This code is used in section 3*.
5. The following parameters can be changed at compile time to extend or reduce DVItype’s capacity.

\( \text{(Constants in the outer block 5*) } \equiv \)

\( \text{max\_fonts} = 500; \quad \{ \text{maximum number of distinct fonts per DVI file} \} \)
\( \text{max\_widths} = 25000; \quad \{ \text{maximum number of different characters among all fonts} \} \)
\( \text{line\_length} = 79; \quad \{ \text{bracketed lines of output will be at most this long} \} \)
\( \text{stack\_size} = 100; \quad \{ \text{DVI files shouldn’t push beyond this depth} \} \)
\( \text{name\_size} = 10000; \quad \{ \text{total length of all font file names} \} \)

This code is used in section 3*.

7. If the DVI file is badly malformed, the whole process must be aborted; DVItype will give up, after issuing an error message about the symptoms that were noticed.

Such errors might be discovered inside of subroutines inside of subroutines, so a procedure called \textit{jump\_out} has been introduced.

\begin{verbatim}
define jump_out ≡ uexit(1)
define abort(#) ≡
   begin writeLn(stderr, #); jump_out;
   end
define bad_dvi(#) ≡ abort(`Bad DVI file:`, #, `!`) 
\end{verbatim}
8* The character set. Like all programs written with the WEB system, DVItype can be used with any character set. But it uses ASCII code internally, because the programming for portable input-output is easier when a fixed internal code is used, and because DVI files use ASCII code for file names and certain other strings.

The next few sections of DVItype have therefore been copied from the analogous ones in the WEB system routines. They have been considerably simplified, since DVItype need not deal with the controversial ASCII codes less than '40 or greater than '176. If such codes appear in the DVI file, they will be printed as question marks.

⟨Types in the outer block 8*⟩≡

ASCII_code = 0..255; { a subrange of the integers }

See also sections 9* and 21.

This code is used in section 3*.

9* The original Pascal compiler was designed in the late 60s, when six-bit character sets were common, so it did not make provision for lower case letters. Nowadays, of course, we need to deal with both upper and lower case alphabets in a convenient way, especially in a program like DVItype. So we shall assume that the Pascal system being used for DVItype has a character set containing at least the standard visible characters of ASCII code ("!" through "~").

Some Pascal compilers use the original name char for the data type associated with the characters in text files, while other Pascals consider char to be a 64-element subrange of a larger data type that has some other name. In order to accommodate this difference, we shall use the name text_char to stand for the data type of the characters in the output file. We shall also assume that text_char consists of the elements chr(first_text_char) through chr(last_text_char), inclusive. The following definitions should be adjusted if necessary.

define text_char ≡ ASCII_code { the data type of characters in text files }
define first_text_char = 0 { ordinal number of the smallest element of text_char }
define last_text_char = 255 { ordinal number of the largest element of text_char }

⟨Types in the outer block 8*⟩+≡

text_file = packed file of text_char;
To prepare these files for input, we reset them. An extension of Pascal is needed in the case of \texttt{tfm\_file}, since we want to associate it with external files whose names are specified dynamically (i.e., not known at compile time). The following code assumes that ‘\texttt{reset}(f, s)$’ does this, when \(f\) is a file variable and \(s\) is a string variable that specifies the file name. If \texttt{eof}(f) is true immediately after \texttt{reset}(f, s)\) has acted, we assume that no file named \(s\) is accessible.

\begin{verbatim}
procedure open_dvi_file;  \{ prepares to read packed bytes in \texttt{dvi\_file} \}
   begin resetbin(dvi_file, dvi_name); cur_loc ← 0;
   end;

procedure open_tfm_file;  \{ prepares to read packed bytes in \texttt{tfm\_file} \}
   var full_name: ↑char;
   begin full_name ← kpse_find_tfm(cur_name);
       if full_name then
           begin tfm_file ← fopen(full_name, FOPEN_RBIN_MODE);
           end
       else begin tfm_file ← nil;
           end;
   end;
\end{verbatim}

If you looked carefully at the preceding code, you probably asked, “What are \texttt{cur\_loc} and \texttt{cur\_name}?” Good question. They’re global variables: \texttt{cur\_loc} is the number of the byte about to be read next from \texttt{dvi\_file}, and \texttt{cur\_name} is a string variable that will be set to the current font metric file name before \texttt{open\_tfm\_file} is called.

\begin{verbatim}
⟨Globals in the outer block 10⟩ +≡
cur_loc: integer;  \{ where we are about to look, in \texttt{dvi\_file} \}
cur_name: ↑char;  \{ external name \}
\end{verbatim}

Finally we come to the routines that are used only if \texttt{random\_reading} is true. The driver program below needs two such routines: \texttt{dvi\_length} should compute the total number of bytes in \texttt{dvi\_file}, possibly also causing \texttt{eof(dvi\_file)} to be true; and \texttt{move\_to\_byte(n)} should position \texttt{dvi\_file} so that the next \texttt{get\_byte} will read byte \(n\), starting with \(n = 0\) for the first byte in the file.

Such routines are, of course, highly system dependent. They are implemented here in terms of two assumed system routines called \texttt{set\_pos} and \texttt{cur\_pos}. The call \texttt{set\_pos}(\(f, n\)) moves to item \(n\) in file \(f\), unless \(n\) is negative or larger than the total number of items in \(f\); in the latter case, \texttt{set\_pos}(\(f, n\)) moves to the end of file \(f\). The call \texttt{cur\_pos}(\(f\)) gives the total number of items in \(f\), if \texttt{eof}(\(f\)) is true; we use \texttt{cur\_pos} only in such a situation.

\begin{verbatim}
function dvi\_length: integer;
   begin xfseek(dvi_file, 0, 2, dvi_name); cur_loc ← xftell(dvi_file, dvi_name); dvi\_length ← cur\_loc;
   end;

procedure move\_to\_byte(n : integer);
   begin xfseek(dvi_file, n, 0, dvi_name); cur\_loc ← n;
   end;
\end{verbatim}
The starting page specification is recorded in two global arrays called `start_count` and `start_there`. For example, ‘1.*−5’ is represented by `start_there[0] = true, start_count[0] = 1, start_there[1] = false, start_there[2] = true, start_count[2] = −5`. We also set `start_vals = 2`, to indicate that count 2 was the last one mentioned. The other values of `start_count` and `start_there` are not important, in this example.

42* The starting page specification is recorded in two global arrays called `start_count` and `start_there`. For example, ‘1.*−5’ is represented by `start_there[0] = true, start_count[0] = 1, start_there[1] = false, start_there[2] = true, start_count[2] = −5`. We also set `start_vals = 2`, to indicate that count 2 was the last one mentioned. The other values of `start_count` and `start_there` are not important, in this example.

\[
\begin{align*}
\text{Globals in the outer block} & \equiv
\end{align*}
\]

\[
\begin{align*}
\text{start_count: array [0 .. 9] of integer; } & \{ \text{count values to select starting page} \} \\
\text{start_there: array [0 .. 9] of boolean; } & \{ \text{is the start_count value relevant?} \} \\
\text{start_vals: 0 .. 9; } & \{ \text{the last count considered significant} \} \\
\text{count: array [0 .. 9] of integer; } & \{ \text{the count values on the current page} \}
\end{align*}
\]

43* Initializations are done sooner now.

45* No dialog.

47* During the dialog, `DVItype` will treat the first blank space in a line as the end of that line. Therefore `input_ln` makes sure that there is always at least one blank space in `buffer`.

48* No dialog.

49* No dialog.

50* No dialog (50).

51* No dialog (51).

52* No dialog (52).

53* No dialog (53).

54* No dialog (54).

55* No dialog (55).
56* After the dialog is over, we print the options so that the user can see what DVI type thought was specified.

\[
\begin{align*}
\text{print}_\text{ln}(\text{Options selected:} \star); & \quad \text{print}(\text{Starting page} = \star) \\text{;} \\
\text{for } k \leftarrow 0 \text{ to start_vals do} \quad & \begin{aligned}
\begin{align*}
\text{begin if } \text{start there}[k] \text{ then } & \text{print}(\text{start_count}[k] : 1) \\
\text{else } & \text{print}(\star) \\
\text{if } k < \text{start_vals then } & \text{print}(\star) \\
\text{else } & \text{\text{print}_\text{ln}(\star)} \\
\end{align*}
\end{aligned} \end{align*}
\]

\[
\text{end} \; ; \text{print}_\text{ln}(\text{Maximum number of pages} = \star, \text{max_pages} : 1) ; \text{print}(\text{Output level} = \star, \text{out_mode} : 1) \; ; \begin{align*}
\text{case } \text{out_mode of} \quad & \begin{aligned}
\begin{align*}
\text{errors-only: } & \text{print}_\text{ln}(\text{showing bops, fonts, and error messages only}) \star) \\
\text{terse: } & \text{print}_\text{ln}(\text{terse}) \star) \\
\text{mnemonics-only: } & \text{print}_\text{ln}(\text{mnemonics}) \star) \\
\text{verbose: } & \text{print}_\text{ln}(\text{verbose}) \star) \\
\text{the works: if random reading then } & \text{print}_\text{ln}(\text{the works}) \star) \\
\text{else begin out_mode } & \leftarrow \text{verbose; print}_\text{ln}(\text{the works: same as level 3 in this DVI type}) \star) \\
\text{end} \end{aligned}
\end{align*}
\end{align*}
\]

\[
\begin{align*}
\text{end} \; ; \text{print}(\text{Resolution} = \star) \; ; \text{print}_\text{real}(\text{resolution}, 12, 8) \; ; \text{print}_\text{ln}(\text{pixels per inch}) \star) \\
\text{if new_mag } > 0 \text{ then} \quad & \begin{aligned}
\begin{align*}
\text{begin } & \text{print}(\text{New magnification factor} = \star) \; ; \text{print}_\text{real}(\text{new_mag}/1000.0, 8, 3) \; ; \text{print}_\text{ln}(\star) \\
\end{align*}
\end{aligned} \end{align*}
\]

This code is used in section 107*.
The following subroutine does the necessary things when a \textit{fnt\_def} command is being processed.

\begin{verbatim}
59* procedure define_font(e: integer); { e is an external font number }
    var f: 0..max\_fonts; p: integer; { length of the area/directory spec }
    n: integer; { length of the font name proper }
    c, q, d, m: integer; { check sum, scaled size, design size, magnification }
    r: 0..name\_size; { current filename length }
    j, k: 0..name\_size; { indices into names }
    mismatch: boolean; { do names disagree? }
    begin if nf = max\_fonts then
        abort(`DVItype\_capacity\_exceeded\(max\_fonts=\', max\_fonts : 1, `) !`);
        font\_num\[nf\] \leftarrow e; f \leftarrow 0;
        while font\_num\[f\] \neq e do incr\(f\);
        ⟨Read the font parameters into position for font nf, and print the font name 61⟩;
        if ((out\_mode = the\_works) \land in\_postamble) \lor ((out\_mode < the\_works) \land \lnot in\_postamble) then
            begin if f < nf then print\_ln(`---this\_font\_was\_already\_defined!`
            end
        else begin if f = nf then print\_ln(`---this\_font\_wasn`\_t\_loaded\_before!`
            end;
        if f = nf then ⟨Load the new font, unless there are problems 62⟩;
        else ⟨Check that the current font definition matches the old one 60⟩;
    end;

62* ⟨Load the new font, unless there are problems 62⟩ ≡
    begin ⟨Move font name into the cur\_name string 66⟩;
        open\_tfm;
        if eof(tfm\_file) then print(`---not\_loaded\(\_\_TFM\_file\_can\_\_t\_be\_opened!`
        else begin if (q \leq 0) \lor (q \geq `1000000000) then print(`---not\_loaded\(\_\_bad\_scale\(\_\_\_q\_1\_\)!
            else if (d \leq 0) \lor (d \geq `1000000000) then print(`---not\_loaded\(\_\_bad\_design\_size\(\_\_d\_1\_\)!
            else if in\_TFM\(q\) then ⟨Finish loading the new font info 63⟩;
        end;
        if out\_mode = errors\_only then print\_ln(`\_\_`)
        if tfm\_file then xfclose(tfm\_file, cur\_name); { should be the kpse\_find\_tfm result }
        free(cur\_name); { We xmalloc`d this before we got called. }
    end
\end{verbatim}

This code is used in section 59*.

64* If \(p = 0\), i.e., if no font directory has been specified, \texttt{DVItype} is supposed to use the default font directory, which is a system-dependent place where the standard fonts are kept. The string variable \texttt{default\_directory} contains the name of this area.

Under Unix, users have a path searched for fonts, there’s no single default directory.

65* (No initialization needs to be done. Keep this module to preserve numbering.)
The string *cur_name* is supposed to be set to the external name of the TFM file for the current font. We do not impose a maximum limit here. It’s too bad there is a limit on the total length of all filenames, but it doesn’t seem worth reprogramming all that.

\[
\text{define } \text{name}_\text{start} \equiv \text{font}\_\text{name}[\text{nf}]
\]
\[
\text{define } \text{name}_\text{end} \equiv \text{font}\_\text{name}[\text{nf} + 1]
\]

(Move font name into the *cur_name* string \(66^\ast\)\) \(\equiv\)
\[
r \leftarrow \text{name}_\text{end} - \text{name}_\text{start}; \ \text{cur}\_\text{name} \leftarrow \text{xmalloc}\_\text{array}(\text{char}, r);
\]
{ *strncpy* might be faster, but it’s probably a good idea to keep the *xchr* translation. }

\[
\text{for } k \leftarrow \text{name}_\text{start} \text{ to } \text{name}_\text{end} \text{ do}
\]
\[
\text{begin } \text{cur}\_\text{name}[k - \text{name}_\text{start}] \leftarrow \text{xchr}\[\text{names}[k]];
\]
\[
\text{end};
\]
\[
\text{cur}\_\text{name}[r] \leftarrow 0; \ \{ \text{Append null byte for C.} \}
\]

This code is used in section \(62^\ast\).
Before we get into the details of `do_page`, it is convenient to consider a simpler routine that computes the first parameter of each opcode.

### Define

```c
#define four_cases(#) \equiv \#, \# + 1, \# + 2, \# + 3
#define eight_cases(#) \equiv four_cases(#), four_cases(# + 4)
#define sixteen_cases(#) \equiv eight_cases(#), eight_cases(# + 8)
#define thirty_two_cases(#) \equiv sixteen_cases(#), sixteen_cases(# + 16)
#define sixty_four_cases(#) \equiv thirty_two_cases(#), thirty_two_cases(# + 32)
```

```c
function first_par(o : eight_bits) : integer;
    begin case o of
        sixty_four_cases(set_char_0), sixty_four_cases(set_char_0 + 64): first_par \leftarrow o - set_char_0;
        set1, put1, fnt1, xxx1, fnt_def1: first_par \leftarrow get_byte;
        set1 + 1, put1 + 1, fnt1 + 1, xxx1 + 1, fnt_def1 + 1: first_par \leftarrow get_two_bytes;
        set1 + 2, put1 + 2, fnt1 + 2, xxx1 + 2, fnt_def1 + 2: first_par \leftarrow get_three_bytes;
        right1, w1, x1, down1, y1, z1: first_par \leftarrow signed_byte;
        right1 + 1, w1 + 1, x1 + 1, down1 + 1, y1 + 1, z1 + 1: first_par \leftarrow signed_pair;
        right1 + 2, w1 + 2, x1 + 2, down1 + 2, y1 + 2, z1 + 2: first_par \leftarrow signed_trio;
        set1 + 3, set_rule, put1 + 3, put_rule, right1 + 3, w1 + 3, x1 + 3, down1 + 3, y1 + 3, z1 + 3, fnt1 + 3,
        xxx1 + 3, fnt_def1 + 3: first_par \leftarrow signed_quad;
        nop, bop, cop, push, pop, pre, post, post_post, undefined_commands: first_par \leftarrow 0;
        w0: first_par \leftarrow w;
        x0: first_par \leftarrow x;
        y0: first_par \leftarrow y;
        z0: first_par \leftarrow z;
        sixty_four_cases(fnt_num_0): first_par \leftarrow o - fnt_num_0;
    othercases abort(‘internal_error’);
endcases;
end;
```
Commands are broken down into “major” and “minor” categories: A major command is always shown in full, while a minor one is put into the buffer in abbreviated form. Minor commands, which account for the bulk of most DVI files, involve horizontal spacing and the typesetting of characters in a line; these are shown in full only if \texttt{out\_mode} \geq \texttt{verbose}.

\begin{verbatim}
define show(#) ≡ begin flush_text; showing ← true; print(a:1, ´:₁:´, #); if show_opcodes ∧ (o \geq 128) then print(´:₁:{´, o:1, ´}:´); end
#define major(#) ≡ if out_mode > errors_only then show(#)
#define minor(#) ≡ if out_mode > terse then begin showing ← true; print(a:1, ´:₁:´, #); if show_opcodes ∧ (o \geq 128) then print(´:₁:{´, o:1, ´}:´); end
#define error(#) ≡ if ¬showing then show(#)
else print(´:₁:´)
\end{verbatim}

(Translate the next command in the DVI file; \texttt{goto} 9999 with \texttt{do\_page} = \texttt{true} if it was \texttt{eop}; \texttt{goto} 9998 if premature termination is needed)

\begin{verbatim}
begin a ← cur_loc; showing ← false; o ← get_byte; p ← first_par(o);
if eof(dvi_file) then bad_dvi(´the file ended prematurely´);
\end{verbatim}

\begin{verbatim}
⟨Start translation of command \textit{o} and \texttt{goto} the appropriate label to finish the job⟩
\end{verbatim}

\begin{verbatim}
⟨Finish a command that either sets or puts a character, then \texttt{goto move\_right} or \texttt{done}⟩
\end{verbatim}

\begin{verbatim}
⟨Finish a command that sets \textit{h} ← \textit{h} + \textit{q}, then \texttt{goto done}⟩
\end{verbatim}

\begin{verbatim}
⟨Finish a command that sets \textit{s}, \textit{h}, \textit{v}, \textit{w}, \textit{x}, \textit{y}, \textit{z}, \textit{hh}, and \textit{vv}; then \texttt{goto done}⟩
\end{verbatim}

\begin{verbatim}
⟨If \texttt{showing} then \texttt{print\_ln(´:₁:´)}⟩
\end{verbatim}

This code is used in section 79.
99* The `scan_bop` procedure reads DVI commands following the preamble or following `eop`, until finding either `bop` or the postamble.

⟨Declare the procedure called `scan_bop` 99⟩ ≡

```latex
procedure scan_bop:
    var k: 0..255;  \{ command code \}
    begin repeat if `eof(dvi_file)` then `bad_dvi(\'the_file\_ended\_prematurely\');
        k ← `get_byte`;
        if (k ≥ \texttt{fnt\_def1}) \& (k < \texttt{fnt\_def1 + 4}) then
            begin `define_font(first_par(k));`
                if `out\_mode ≠ errors\_only` then `print\_ln(\'\;\);`
            end;
        end;
    until k ≠ `nop`;
    if k = `post` then `in\_postamble ← true`
else begin if k ≠ `bop` then `bad_dvi(\'byte\_\_cur\_loc - 1 : 1, \'is\_not\_bop\');`
        `new\_backpointer ← cur\_loc - 1; incr(page\_count);`
        for k ← 0 to 9 do `count[k] ← signed\_quad;`
        if `signed\_quad ≠ old\_backpointer` then
            `print\_ln(\'backpointer\_in\_byte\_cur\_loc - 4 : 1, \'should\_be\_, old\_backpointer : 1, \'!\');`
        `old\_backpointer ← new\_backpointer;`
    end;
end;
```

This code is used in section 95.
§107* The main program. Now we are ready to put it all together. This is where DVItype starts, and where it ends.

\[
\text{begin \ initialize: \ \{ \ get \ all \ variables \ initialized \}\;}
\]
\[
\langle \ \text{Process \ the \ preamble \ 109}\rangle\;}
\]
\[
\text{if \ out\_mode = \ the\_works \ then \ \{ \ \text{random\_reading} = \true \}\;}
\]
\[
\langle \ \text{Find \ the \ postamble, \ working \ back \ from \ the \ end \ 100}\rangle\;}
\]
\[
\text{in\_postamble} \leftarrow \true; \ \text{read\_postamble}; \ \text{in\_postamble} \leftarrow \false;\]
\[
\langle \ \text{Count \ the \ pages \ and \ move \ to \ the \ starting \ page \ 102}\rangle\;}
\]
\[
\text{end};\]
\[
\text{skip\_pages(\false);}\]
\[
\text{if \ \neg \ \text{in\_postamble} \ then \ \langle \ \text{Translate \ up \ to \ max\_pages \ pages \ 111}\rangle;}\]
\[
\text{if \ out\_mode < \the\_works \ then \ \begin{cases} \text{if \ \neg \ \text{in\_postamble} \ then \ \text{skip\_pages(\true);} & \text{if \ signed\_quad} \neq \old\_backpointer \ then \end{cases} \}
\]
\[
\text{print\_ln(`}\text{backpointer}_{\text{in\_byte}}, \text{cur\_loc} - 4 : 1, `\text{should\_be}, \old\_backpointer : 1, `!`);}\]
\[
\text{read\_postamble};\]
\[
\text{end};\]
\[
\text{end.}\]

110* The conversion factor \(conv\) is figured as follows: There are exactly \(\frac{n}{d}\) decimicrons per DVI unit, and 254000 decimicrons per inch, and \(resolution\) pixels per inch. Then we have to adjust this by the stated amount of magnification.

\[
\langle \ \text{Compute \ the \ conversion \ factors \ 110*} \rangle \equiv
\]
\[
\text{numerator} \leftarrow \text{signed\_quad}; \ \text{denominator} \leftarrow \text{signed\_quad};\]
\[
\text{if} \ \text{numerator} \leq 0 \ \text{then} \ \text{bad\_dvi(`\text{numerator} is}, \text{numerator : 1});\]
\[
\text{if} \ \text{denominator} \leq 0 \ \text{then} \ \text{bad\_dvi(`\text{denominator} is}, \text{denominator : 1});\]
\[
\text{print\_ln(`\text{numerator/denominator=}, \text{numerator : 1, `!/}, \text{denominator : 1});}\]
\[
\text{tfm\_conv} \leftarrow (25400000.0/\text{numerator}) \cdot (\text{denominator}/473628672)/16.0;\]
\[
\text{conv} \leftarrow (\text{numerator}/254000.0) \cdot (\text{resolution}/\text{denominator}); \ \text{mag} \leftarrow \text{signed\_quad};\]
\[
\text{if} \ \text{new\_mag} > 0 \ \text{then} \ \text{mag} \leftarrow \text{new\_mag}\]
\[
\text{else if} \ \text{mag} \leq 0 \ \text{then} \ \text{bad\_dvi(`\text{magnification} is}, \text{mag : 1});\]
\[
\text{true\_conv} \leftarrow \text{conv}; \ \text{conv} \leftarrow \text{true\_conv} \cdot (\text{mag}/1000.0); \ \text{print(`\text{magnification=}, \text{mag : 1, `});}\]
\[
\text{print\_real(\text{conv},16,8); \ \text{print\_ln(`\text{pixels per DVI unit})}\]
\]

This code is used in section 109.
112* System-dependent changes. Parse a Unix-style command line.

```c
define argument_is(#) ≡ (strcmp(long_options[option_index], name, #) = 0)
(Define parse_arguments 112*) ≡
procedure parse_arguments;
  const n_options = 8; {Pascal won’t count array lengths for us.}
  var long_options: array [0..n_options] of getopt_struct;
  getopt_return_val: integer; option_index: c_int_type; current_option: 0..n_options; end_num: ↑char;
  {for page-start}
begin {Define the option table 113*};
repeat getopt_return_val ← getopt_long_only(argc, argv, ´`, long_options, address_of (option_index));
  if getopt_return_val = −1 then
    begin do nothing; {End of arguments; we exit the loop below.}
    end
  else if argument_is(`help`) then
    begin usage(my_name);
    end
  else if argument_is(`version`) then
    begin print_version_and_exit(banner, nil, `D.E. Knuth`, nil);
    end
  else if argument_is(`output-level`) then
    begin if (optarg[0] < `0`) ∨ (optarg[0] > `4`) ∨ (optarg[1] ≠ 0) then
      begin write ln(stderr, `Value for `−−output-level must be >= `0` and <= `4`.`);
      uexit(1);
      end;
      out_mode ← optarg[0] − `0`;
    end
  else if argument_is(`page-start`) then
    begin {Determine the desired start_count values from optarg 117*};
    end
  else if argument_is(`max-pages`) then
    begin max_pages ← atou(optarg);
    end
  else if argument_is(`dpi`) then
    begin resolution ← atof(optarg);
    end
  else if argument_is(`magnification`) then
    begin new_mag ← atou(optarg);
    end; {Else it was a flag; getopt has already done the assignment.}
until getopt_return_val = −1; {Now optind is the index of first non-option on the command line.}
if (optind + 1 ≠ argc) then
  begin write ln(stderr, my_name, `:` Need exactly one file argument.`); usage(my_name);
  end;
  dvi_name ← extend_filename(cmdline(optind), `dvi`);
end;
```

This code is used in section 3*. 
113* Here are the options we allow. The first is one of the standard GNU options.

(Define the option table 113*) \equiv
\begin{align*}
current\_option & \leftarrow 0; \ long\_options[\text{current}\_option].\text{name} \leftarrow \text{\texttt{help}}; \\
& long\_options[\text{current}\_option].\text{has\_arg} \leftarrow 0; \ long\_options[\text{current}\_option].\text{flag} \leftarrow 0; \\
& long\_options[\text{current}\_option].\text{val} \leftarrow 0; \ incr(\text{current}\_option);
\end{align*}

See also sections 114*, 115*, 116*, 118*, 119*, 120*, 121*, and 123*.

This code is used in section 112*.

114* Another of the standard options.

(Define the option table 113*) \equiv
\begin{align*}
& long\_options[\text{current}\_option].\text{name} \leftarrow \text{\texttt{version}}; \ long\_options[\text{current}\_option].\text{has\_arg} \leftarrow 0; \\
& long\_options[\text{current}\_option].\text{flag} \leftarrow 0; \ long\_options[\text{current}\_option].\text{val} \leftarrow 0; \ incr(\text{current}\_option);
\end{align*}

115* How verbose to be.

(Define the option table 113*) \equiv
\begin{align*}
& long\_options[\text{current}\_option].\text{name} \leftarrow \text{\texttt{output\_level}}; \ long\_options[\text{current}\_option].\text{has\_arg} \leftarrow 1; \\
& long\_options[\text{current}\_option].\text{flag} \leftarrow 0; \ long\_options[\text{current}\_option].\text{val} \leftarrow 0; \ incr(\text{current}\_option); \\
& \text{out\_mode} \leftarrow \text{\texttt{the\_works}}; \ \{ \text{default} \}
\end{align*}

116* What page to start at.

(Define the option table 113*) \equiv
\begin{align*}
& long\_options[\text{current}\_option].\text{name} \leftarrow \text{\texttt{page\_start}}; \ long\_options[\text{current}\_option].\text{has\_arg} \leftarrow 1; \\
& long\_options[\text{current}\_option].\text{flag} \leftarrow 0; \ long\_options[\text{current}\_option].\text{val} \leftarrow 0; \ incr(\text{current}\_option);
\end{align*}

117* Parsing the starting page specification is a bit complicated.

(Determine the desired start\_count values from optarg 117*) \equiv
\begin{align*}
k & \leftarrow 0; \ \{ \text{which } \texttt{\count} \text{ register we're on} \} \\
m & \leftarrow 0; \ \{ \text{position in optarg} \} \\
\text{while optarg}[m] \text{ do} \\
& \begin{align*}
& \text{begin if optarg}[m] = \text{\texttt{*}} \text{ then} \\
& & \begin{align*}
& & \text{begin start\_there}[k] \leftarrow \text{false}; \ incr(m); \\
& & \text{end}
& \end{align*}
& \end{align*}
& \text{else if optarg}[m] = \text{\texttt{.}} \text{ then} \\
& & \begin{align*}
& & \text{begin incr(k);} \\
& & \text{if } k \geq 10 \text{ then} \\
& & & \begin{align*}
& & & \text{begin write\_ln(stderr, my\_name, \texttt{"\$More than ten\_count\_registers\_specified."});} \\
& & & \text{uexit(1);} \\
& & & \text{end;}
& & & \text{incr(m);} \\
& & & \text{end}
& & \end{align*}
& \end{align*}
& \text{else begin start\_count}[k] \leftarrow \text{strtol(optarg + m, address\_of(\text{end\_num}), 10);}
& \text{if end\_num = optarg + m then} \\
& & \begin{align*}
& & \text{begin write\_ln(stderr, my\_name, \texttt{"\$\_page\_start\_values\_must\_be\_numeric\_or\_\texttt{.}.* \});} \\
& & \text{uexit(1);} \\
& & \text{end;}
& & \text{start\_there}[k] \leftarrow \text{true; m} \leftarrow m + \text{end\_num} - (\text{optarg + m);}
& & \text{end}
& \end{align*}
& \text{end;}
& \text{start\_vals} \leftarrow k;
\end{align*}

This code is used in section 112*.
118*  How many pages to do.
(Define the option table 113*) +≡

\[
\begin{align*}
&\text{long_options[\text{current_option}].name} \leftarrow \text{`max-pages`};
&\text{long_options[\text{current_option}].has_arg} \leftarrow 1;
&\text{long_options[\text{current_option}].flag} \leftarrow 0;
&\text{long_options[\text{current_option}].val} \leftarrow 0;
&\text{incr(\text{current_option})};
&\text{max_pages} \leftarrow 1000000; \\
&\{\text{default}\}
\end{align*}
\]

119*  Resolution, in pixels per inch.
(Define the option table 113*) +≡

\[
\begin{align*}
&\text{long_options[\text{current_option}].name} \leftarrow \text{`dpi`};
&\text{long_options[\text{current_option}].has_arg} \leftarrow 1;
&\text{long_options[\text{current_option}].flag} \leftarrow 0;
&\text{long_options[\text{current_option}].val} \leftarrow 0;
&\text{incr(\text{current_option})};
&\text{resolution} \leftarrow 300.0; \\
&\{\text{default}\}
\end{align*}
\]

120*  Magnification to apply.
(Define the option table 113*) +≡

\[
\begin{align*}
&\text{long_options[\text{current_option}].name} \leftarrow \text{`magnification`};
&\text{long_options[\text{current_option}].has_arg} \leftarrow 1;
&\text{long_options[\text{current_option}].flag} \leftarrow 0;
&\text{long_options[\text{current_option}].val} \leftarrow 0;
&\text{incr(\text{current_option})};
&\text{new_mag} \leftarrow 0; \\
&\{\text{default is to keep the old one}\}
\end{align*}
\]

121*  Whether to show numeric opcodes.
(Define the option table 113*) +≡

\[
\begin{align*}
&\text{long_options[\text{current_option}].name} \leftarrow \text{`show-opcodes`};
&\text{long_options[\text{current_option}].has_arg} \leftarrow 0;
&\text{long_options[\text{current_option}].flag} \leftarrow \text{address_of(\text{show_opcodes})};
&\text{long_options[\text{current_option}].val} \leftarrow 1;
&\text{incr(\text{current_option})};
\end{align*}
\]

122*  (Globals in the outer block 10) +≡

\[
\begin{align*}
&\text{show_opcodes}: \text{c_int_type};
\end{align*}
\]

123*  An element with all zeros always ends the list.
(Define the option table 113*) +≡

\[
\begin{align*}
&\text{long_options[\text{current_option}].name} \leftarrow 0; \\
&\text{long_options[\text{current_option}].has_arg} \leftarrow 0;
&\text{long_options[\text{current_option}].flag} \leftarrow 0; \\
&\text{long_options[\text{current_option}].val} \leftarrow 0;
\end{align*}
\]

124*  Global filenames.
(Globals in the outer block 10) +≡

\[
\begin{align*}
&\text{dvi_name}: \text{const c_string};
\end{align*}
\]
125* Index. Pointers to error messages appear here together with the section numbers where each identifier is used.

The following sections were changed by the change file: 1, 3, 4, 5, 7, 8, 9, 23, 24, 28, 42, 43, 45, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 59, 62, 64, 65, 66, 75, 80, 99, 107, 110, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125.

- dpi: 119*
- help: 113*
- magnification: 120*
- max-pages: 118*
- output-level: 115*
- page-start: 116*
- show-opcodes: 121*
- version: 114*

a: 27, 79, 82.
abort: 7* 59*, 61, 75*, 102.
abs: 63, 73, 85, 91, 92.
address_of: 112*, 117*, 121*
after_pre: 101, 102, 109.
all 223s: 100.
alpha: 34, 37, 38.
argc: 112*
argument_is: 112*
argv: 3*, 112*
arithmetic overflow...: 91, 92.
ASCII_code: 8*, 9*, 10, 30, 67, 70.
atof: 112*
atou: 119*
b: 27.
backpointer...should be p: 99*, 107*
bad design size: 62*
Bad DVI file: 7*
bad postamble pointer: 105.
bad scale: 62*
bad_char: 82, 87.
bad_dvi: 7*, 80*, 96, 99*, 100, 102, 105, 109, 110* 111.
banner: 1*, 3*, 112*
beta: 34, 37, 38.
beware: check sums do not agree: 63.
beware: design sizes do not agree: 63.
boolean: 34, 42*, 44, 57, 59*, 78, 79, 82, 95, 97.
bop occurred before eop: 83.
bop_seen: 95.
break: 46.
Breitenlohner, Peter: 1*
buffer: 47*
byte n is not bop: 99*, 102.
byte n is not post: 100.
byte n is not postpost: 106.
byte_file: 21*, 22.
b0: 25, 26, 35, 36, 37.
b1: 25, 26, 35, 37.
b2: 25, 26, 35, 37.
b3: 25, 26, 35, 37.
c: 27, 59*
c_int_type: 112*, 122*
change_font: 77, 82, 86.
char: 9*, 23*, 24*, 60*, 112*
char_pixel_width: 39, 89.
char_width: 30, 39, 89.
char_width_end: 30, 39.
character c invalid...: 89.
check sum: 18.
check sum doesn’t match: 60.
check sums do not agree: 63.
Chinese characters: 15, 89.
chr: 9*, 10, 12.
cmdline: 112*
const_c_string: 124*
conv: 39, 40, 61, 63, 76, 110*
count: 42*, 44, 99*, 102, 111.
cur_font: 77, 78, 79, 84, 85, 89, 94.
cur_name: 23*, 24*, 62*, 66*
cur_pos: 28*
d: 27, 59*
decl: 6, 83, 96, 100, 109, 111.
deepener than claimed...: 83.
default_directory: 64*
den: 15, 17, 19.
denominator: 39, 103, 110*
denominator doesn’t match: 103.
denominator is wrong: 110*
design size doesn’t match: 60.
design sizes do not agree: 63.
do_nothing: 6, 96, 112*
do_page: 71, 75*, 77, 78, 79, 81, 83, 95, 111.
done: 4*, 79, 80*, 81, 82, 83, 86, 87, 89, 90, 91, 92, 93, 94, 111.
down_the_drain: 95, 96.
down1: 15, 16, 75*, 85.
down2: 15.
down3: 15.
down4: 15.
DVI files: 13.
DVITYPE_HELP: 112*
DVIType capacity exceeded...: 59*61.
DVIType needs larger...: 35.
e: 59*
eight_bits: 21, 25, 27, 75*79, 82.
eight_cases: 75*
eop: 13, 15, 16, 18, 41, 75*83, 96, 99*.
error: 80*82, 83, 87, 89, 91, 92, 94.
errors_only: 41, 56*62*69, 80*99*.
extend_filename: 112*.
false: 2, 20, 34, 42*44, 58, 60, 77, 79, 80*82, 87, 95, 98, 103, 107*117*.
fin_rule: 77, 79, 80*81.
fin_set: 77, 79, 80*81, 88.
First byte isn’t...: 109.
first_backpointer: 100, 101, 102.
first_par: 75*80*81, 96, 99*106.
first_text_char: 9*, 12.
fix_word: 37.
flag: 113*114*115*116*118*119*120*121*123*.
flnlash_text: 69, 70, 80*.
fln_def1: 15, 16, 75*86, 96, 99*106.
fln_def2: 15.
fln_def3: 15.
fln_def4: 15.
fln_num_0: 15, 16, 75*86.
fln_num_1: 15.
fln_num_63: 15.
fln: 15, 16, 75*86.
fln2: 15.
fln3: 15.
fln4: 15.
font name doesn’t match: 60.
font bc: 30, 31, 35, 40, 89.
font_check_type: 30, 60, 61.
font_design_size: 30, 60, 61.
font_ec: 30, 31, 35, 89.
font_name: 30, 31, 32, 60, 61, 66*.
font_num: 30, 59*94.
font_scaled_size: 30, 60, 61.
font_space: 30, 31, 63, 84, 85.
ofnopen: 23*.
FOPEN_RB instant: 23*.  

four_cases: 75*81, 82, 84, 85, 86, 96.
free: 62*.
Fuchs, David Raymond: 1*13, 20.
full_name: 23*.
get_byte: 27, 28*61, 75*80*87, 96, 99*100, 102, 105, 106, 109.
get_three_bytes: 27, 75*.
get_two_bytes: 27, 75*103.
gotopt: 112*.
gotopt_long_only: 112*.
gotopt_return_val: 112*.
gotopt_struct: 112*.
h: 72.
has_arg: 113*114*115*116*118*119*120*121*123*.
hh: 72, 79, 83, 84, 89, 90, 91, 93.
hhh: 79, 91.
hhstack: 72, 83.
hstack: 72, 83.
i: 3*17.
ID byte is wrong: 100.
id_byte: 17, 100, 105, 109.
identification...should be n: 105, 109.
illegal command at byte n: 96.
in_postamble: 57, 58, 59*95, 99*102, 107*111.
in_TFM: 34, 37, 62*.
in_width: 33, 37, 40.
inr: 6, 28, 27, 59*60, 63, 70, 83, 94, 99*102, 113*.
inr4: 114*115*116*117*118*119*120*121*.
infinity: 91, 92.
initialize: 3*107*.
input_ext: 47*.
integer: 3*21, 24*27, 28*30, 32, 33, 34, 39, 41, 42*59*72, 73, 75*76, 78, 79, 82, 95, 97, 101, 103, 108, 112*.
invalid_font: 30, 31, 32, 79, 89, 94.
invalid_width: 30, 40, 89.
j: 59*.
Japanese characters: 15, 89.
jump_out: 7*.
k: 17, 32, 34, 44, 59*69, 82, 95, 99*103, 108.
kpse_find_tfm: 23*62*.
kpse_set_program_name: 3*.
last_text_char: 9*12.
lh: 34, 35.
line_length: 5*67, 69, 70.
long_options: 112*113*114*115*116*118*119*.
120*121*123*.
m: 59*103, 108.
mag: 15, 17, 18, 19, 39, 103, 110*.
magnification doesn’t match: 103.
magnification is wrong: 110*.
§125  DVI type changes for C

match:  44.
max_drift:  91, 92.
max_fonts:  5*, 30, 59.*
max_h:  73, 74, 91, 103, 104.
max_pages:  41, 56*, 111, 112*, 118.*
max_s:  73, 74, 83, 103, 104.
max_s_so_far:  73, 74, 83, 104.
max_v:  73, 74, 92, 103, 104.
max_v_so_far:  73, 74, 92, 104.
max_widths:  5*, 30, 34, 35, 39.
minor:  80*, 83, 84, 88.
mismatch:  59*, 60.
mnemonics_only:  41, 56*, 90, 91, 92, 93, 94.
move_down:  77, 82, 85.
move_right:  77, 79, 80*, 84, 89, 90.
move_to_byte:  28*, 100, 102.
my_name:  1*, 3*, 112*, 117*.
n:  59*, 70, 108.
n_options:  112*.
name_end:  66*.
nname_start:  66*.
names:  30, 32, 59*, 60, 61, 66*.
new_backpointer:  97, 99*.
new_mag:  41, 56*, 103, 110*, 112*, 120*.
nf:  30, 31, 35, 40, 59*, 60, 61, 63, 66*, 94.
non-ASCII character...:  87.
nop:  13, 15, 16, 18, 19, 75*, 83, 99*, 106.
not enough signature bytes...:  105.
null font name:  61.
num:  15, 17, 19.
numerator:  39, 103, 110*.
numerator doesn't match:  103.
numerator is wrong:  110*.

nw:  34, 35, 36, 37.
α:  79, 82.

observed maxh was x:  104.
observed maxstackdepth was x:  104.
observed maxv was x:  104.
old_backpointer:  97, 98, 99*, 102, 107*.
only n bytes long:  100.
optary:  112*, 117*.
optind:  112*.
option_index:  112*.
Options selected:  56*.
ord:  10.

oriental characters:  15, 89.

othercases:  2.
others:  2.
out_mode:  41, 56*, 57, 59*, 62*, 69, 80*, 90, 91, 92,
103, 104, 107*, 112*, 115*.
out_space:  31, 84.
out_text:  70, 84, 88.
out_uneval:  31, 85.
output:  3*.
p:  59*, 79, 82, 95, 103, 108.

page ended unexpectedly:  111.

page link wrong...:  102.
page_count:  73, 74, 99*, 102, 104.
parsen_arguments:  3*, 112*.
pixel_round:  40, 72, 84, 85, 91, 92.
pixel_width:  39, 40.
pop:  14, 15, 16, 19, 75*, 83.
post:  13, 15, 16, 19, 20, 75*, 82, 96, 99*, 100,
101, 102, 103.
post pointer is wrong:  100.
post_loc:  100, 101, 102, 103, 105.
post_post:  15, 16, 19, 20, 75*, 82, 96, 105, 106.
preamble command within a page:  82.
Postamble starts at byte n:  103.
pref:  13, 15, 16, 75*, 82, 96, 109.
preamble command within a page:  82.
print:  3*, 32, 50*, 61, 62*, 63, 69, 80*, 87, 89, 90, 91,
92, 93, 94, 103, 109, 110*, 111.
print_font:  32, 61, 89, 94.
print_in:  3*, 34, 35, 56*, 59*, 60, 62*, 63, 69, 79,
80*, 83, 90, 93, 96, 99*, 102, 103, 104, 105,
print_real:  56*, 110*.
print_version_and_exit:  112*.
pure:  82.
push:  5*, 14, 15, 16, 19, 75*, 83.
push deeper than claimed...:  83.
put_rule:  15, 16, 75*, 81, 90, 96.
put1:  15, 16, 75*, 81, 89.
put2: 15.
put3:  15.
put4:  15.
q:  59*, 79, 82, 103, 108.
r:  59*.
random_reading:  2, 20, 28*, 41, 56*, 100, 107*.
read:  26, 27.
read_postamble:  103, 107*.
read_tfm_word:  26, 35, 36, 37.
real:  33, 39, 41.
reset:  23*.
resetbin:  23*.
right1: 15, 16, 75, 84.
right2: 15.
right3: 15.
right4: 15.
round: 35, 40, 61, 63.
rule_pixels: 15, 76, 90.
s: 78.
scaled: 61.
scaled size doesn’t match: 60.
scan_hop: 95, 99, 111.
set_char_0: 15, 16, 75, 81.
set_char_1: 15.
set_char_127: 15.
set_pos: 28.
set_rule: 13, 15, 16, 75, 81, 96.
sel: 15, 16, 75, 81.
set2: 15.
set3: 15.
set4: 15.
show: 80.
show_opcodes: 80, 121, 122.
show_state: 77, 79, 80, 83.
showing: 61, 78, 80, 87, 90, 91, 92, 93, 94, 95, 103.
signature...should be:...: 105.
signed_byte: 27, 75.
signed_pair: 27, 75.
signed_quad: 27, 61, 75, 90, 96, 99, 100, 102, 103, 105, 107, 110.
signed_trio: 27, 75.
sixteen_cases: 75.
sixty_four_cases: 75, 86.
skip_pages: 95, 107.
sp: 17.
special_cases: 78, 81, 82.
ss: 78, 83, 93.
stack not empty:...: 83.
stack_size: 5, 72, 74, 83.
start_count: 42, 44, 56, 117.
start_loc: 101, 102.
start_match: 44, 95, 102.
start_val: 42, 44, 56, 117.
start_vals: 42, 44, 56, 111, 117.
start: 95, 97, 98.
starting page number:...: 102.
stderr: 7, 112, 117.
stdout: 3.
strcmp: 112.
string of negative length: 87.
strncpy: 66.
strtol: 117.

system dependencies: 2, 7, 9, 20, 21, 23, 26, 27, 28, 40, 41, 46, 47, 64, 66.
term_out: 46.
terse: 41, 56, 80.
text_buf: 67, 69, 70.
text_char: 9, 10.
text_file: 9.
text_ptr: 67, 68, 69, 70.
TFM files: 29.

TFM file can’t be opened: 62.
TFM file is bad: 34.
tfm_check_sum: 33, 35, 63.
tfm_conv: 33, 35, 110.
tfm_design_size: 33, 35, 63.

the file ended prematurely: 80, 96, 99.

there are really n pages: 102, 104.
thirty_two_cases: 75.
this font is magnified: 63.
this font was already defined: 59.

this font wasn’t loaded before: 59.
total_pages: 73, 102, 103, 104.
true: 2, 28, 34, 42, 44, 60, 79, 80, 82, 83, 87, 95, 99, 100, 102, 107, 112.
true_conv: 39, 61, 63, 110.
trunc: 76.
uexit: 7, 112, 117.

UNDEFINED: 32.

undefined command: 82.
undefined_commands: 16, 75, 96.
update_terminal: 46.
usage: 112.
usage_help: 112.
v: 72.
verbose: 41, 56, 80.
version_string: 3.

vstack: 72, 83.
vv: 72, 79, 83, 85, 92, 93.
vvstack: 72, 83.
vvv: 82, 92.
w: 72.

warning: |h|...: 91.

warning: |v|...: 92.

warning: observed maxh...: 104.
warning: observed maxstack...: 104.
warning: observed maxv...: 104.

width: 30, 36, 39, 40.
width_base: 30, 39, 40.
width_ptr: 30, 31, 34, 35, 36, 40.
wp: 34, 35, 36, 40.
write: 3.
write_in: 3, 7, 112, 117.


\textit{wstack}: 72, 83.
\textit{w0}: 15, 16, 75* 84.
\textit{w1}: 15, 16, 75* 84.
\textit{w2}: 15.
\textit{w3}: 15.
\textit{w4}: 15.
\textit{x}: 17, 72.
\textit{wchar}: 10, 11, 12, 32, 66* 69, 87, 109.
\textit{xfclose}: 62*.
\textit{xfflush}: 28*.
\textit{xftell}: 28*.
\textit{xmalloc}: 62*.
\textit{xmalloc_array}: 66*.
\textit{xord}: 10, 12.
\textit{xstack}: 72, 83.
\textit{xxx1}: 15, 16, 75* 82, 96.
\textit{xxx2}: 15.
\textit{xppxr}: 15.
\textit{xppx4}: 15, 16.
\textit{x0}: 15, 16, 75* 84.
\textit{x1}: 15, 16, 75* 84.
\textit{x2}: 15.
\textit{x3}: 15.
\textit{x4}: 15.
\textit{y}: 72.
\textit{ystack}: 72, 83.
\textit{y0}: 15, 16, 75* 85.
\textit{y1}: 15, 16, 75* 85.
\textit{y2}: 15.
\textit{y3}: 15.
\textit{y4}: 15.
\textit{z}: 34, 72.
\textit{zstack}: 72, 83.
\textit{z0}: 15, 16, 75* 85.
\textit{z1}: 15, 16, 75* 85.
\textit{z2}: 15.
\textit{z3}: 15.
\textit{z4}: 15.
(Cases for commands \texttt{nop, bop, \ldots, pop} \cite{83}) Used in section \ref{81}.
(Cases for fonts \cite{86}) Used in section \ref{82}.
(Cases for horizontal motion \cite{84}) Used in section \ref{81}.
(Cases for vertical motion \cite{85}) Used in section \ref{82}.
(\text{Check that the current font definition matches the old one} \cite{60}) Used in section \ref{59}.
(Compare the \texttt{lуст} parameters with the accumulated facts \cite{104}) Used in section \ref{103}.
(\text{Compute the} \texttt{lуст} \text{parameters} \cite{110} \text{ Used in section} \ref{109}.
(\text{Constants in the outer block} \cite{5*, 82}) Used in section \ref{3}.
(\text{Count the pages and move to the starting page} \cite{102} \text{ Used in section} \ref{107}.
(\text{Declare the function called} \texttt{special_cases} \cite{82} \text{Used in section} \ref{79}.
(\text{Declare the procedure called} \texttt{scan_bop} \cite{99} \text{Used in section} \ref{95}.
(\text{Define the option table} \cite{113*, 114*, 115*, 116*, 118*, 119*, 120*, 121*, 123*} \text{Used in section} \ref{112}.
(\text{Define} \texttt{parse_arguments} \cite{112} \text{Used in section} \ref{3}.
(\text{Determine the desired} \texttt{start_count} \text{values from} \texttt{optarg} \cite{117} \text{Used in section} \ref{112}.
(\text{Find the postamble, working back from the end} \cite{100} \text{Used in section} \ref{107}.
(\text{Finish a command that changes the current font, then} \texttt{goto done} \cite{94} \text{Used in section} \ref{82}.
(\text{Finish a command that either sets or puts a character, then} \texttt{goto move_right or done} \cite{89} \text{Used in section} \ref{80}.
(\text{Finish a command that either sets or puts a rule, then} \texttt{goto move_right or done} \cite{90} \text{Used in section} \ref{80}.
(\text{Finish a command that sets} \texttt{h} \leftarrow \texttt{h} + \texttt{q}, then} \texttt{goto done} \cite{91} \text{Used in section} \ref{80}.
(\text{Finish a command that sets} \texttt{v} \leftarrow \texttt{v} + \texttt{p}, then} \texttt{goto done} \cite{92} \text{Used in section} \ref{82}.
(\text{Finish loading the new font info} \cite{63} \text{Used in section} \ref{62}.
(\text{Globals in the outer block} \cite{10, 22, 24*, 25, 30, 33, 39, 41, 42*, 57, 67, 72, 73, 78, 82, 97, 101, 108, 122*, 124*} \text{Used in section} \ref{80}.
(\text{Labels in the outer block} \cite{4*} \text{Used in section} \ref{3}.
(\text{Load the new font, unless there are problems} \cite{62} \text{Used in section} \ref{59}.
(\text{Make sure that the end of the file is well-formed} \cite{105} \text{Used in section} \ref{103}.
(\text{Move font name into the} \texttt{cur_name} \text{string} \cite{66} \text{Used in section} \ref{62}.
(\text{Move the widths from} \texttt{in_width} \text{to} \texttt{width}, and append \texttt{pixel_width} \text{values} \cite{40} \text{Used in section} \ref{34}.
(\text{Print all the selected options} \cite{56} \text{Used in section} \ref{107}.
(\text{Process the font definitions of the postamble} \cite{106} \text{Used in section} \ref{103}.
(\text{Process the preamble} \cite{109} \text{Used in section} \ref{107}.
(\text{Read and convert the width values, setting up the} \texttt{in_width} \text{table} \cite{37} \text{Used in section} \ref{34}.
(\text{Read past the header data;} \texttt{goto 9997} \text{if there is a problem} \cite{35} \text{Used in section} \ref{34}.
(\text{Read the font parameters into position for font} \texttt{nf}, and print the font name \cite{61} \text{Used in section} \ref{59}.
(\text{Replace} \texttt{z} \text{by} \texttt{z}' \text{and compute} \texttt{\alpha, \beta} \cite{38} \text{Used in section} \ref{37}.
(\text{Set initial values} \cite{11, 12, 31, 58, 68, 74, 98} \text{Used in section} \ref{3}.
(\text{Show the values of} \texttt{ss, h, v, w, x, y, z, hh, and vv}; \text{then} \texttt{goto done} \cite{93} \text{Used in section} \ref{80}.
(\text{Skip until finding} \texttt{eop} \cite{96} \text{Used in section} \ref{95}.
(\text{Start translation of command} \texttt{o} \text{and} \texttt{goto} \text{the appropriate label to finish the job} \cite{81} \text{Used in section} \ref{80}.
(\text{Store character-width indices at the end of the} \texttt{width} \text{table} \cite{36} \text{Used in section} \ref{34}.
(\text{Translate a} \texttt{set_char} \text{command} \cite{88} \text{Used in section} \ref{81}.
(\text{Translate an} \texttt{xxx} \text{command and} \texttt{goto done} \cite{87} \text{Used in section} \ref{82}.
(\text{Translate the next command in the} \texttt{DVI} \text{file;} \texttt{goto 9999} \text{with} \texttt{do_page = true} \text{if it was} \texttt{eop}; \texttt{goto 9998} \text{if premature termination is needed} \cite{80} \text{Used in section} \ref{79}.
(\text{Translate up to} \texttt{max_pages} \text{pages} \cite{111} \text{Used in section} \ref{107}.
(\text{Types in the outer block} \cite{8*, 9*, 21} \text{Used in section} \ref{3}.}