The TFtoPL processor

(Version 3.3, January 2014)
1* Introduction. The TFtoPL utility program converts \TeX{} font metric ("TFM") files into equivalent property-list ("PL") files. It also makes a thorough check of the given TFM file, using essentially the same algorithm as \TeX{}. Thus if \TeX{} complains that a TFM file is "bad," this program will pinpoint the source or sources of badness. A PL file output by this program can be edited with a normal text editor, and the result can be converted back to TFM format using the companion program PLtoTF.

The first TFtoPL program was designed by Leo Guibas in the summer of 1978. Contributions by Frank Liang, Doug Wyatt, and Lyle Ramshaw also had a significant effect on the evolution of the present code.

Extensions for an enhanced ligature mechanism were added by the author in 1989.

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

\begin{verbatim}
define my_name ≡ 'tftopl'
define banner ≡ 'This is TFtoPL, Version 3.3'  { printed when the program starts }
\end{verbatim}

2* This program is written entirely in standard Pascal, except that it occasionally has lower case letters in strings that are output. Such letters can be converted to upper case if necessary. The input is read from tfm file, and the output is written on pl file; error messages and other remarks are written on the output file, which the user may choose to assign to the terminal if the system permits it.

The term print is used instead of write when this program writes on the output file, so that all such output can be easily deflected.

\begin{verbatim}
define print(#) ≡ write(stderr, #)
define print_ln(#) ≡ writeLn(stderr, #)
\end{verbatim}

3* If the program has to stop prematurely, it goes to the 'final_end'.

\begin{verbatim}
define final_end = 9999  { label for the end of it all }
\end{verbatim}

4* The following parameters can be changed at compile time to extend or reduce TFtoPL's capacity.

\begin{verbatim}
⟨Constants in the outer block 4*⟩≡
lig_size = 32510;  { maximum length of lig/kern program, in words (< 2^{15}) }
hash_size = 32579;
\end{verbatim}

See also section 108*.

This code is used in section 2*.
On some systems you may have to do something special to read a packed file of bytes. With C under Unix, we just open the file by name and read characters from it.

\[
\text{Set initial values}\]
\[
\text{tfm\_file} \leftarrow \text{kpse\_open\_file(tfm\_name, kpse\_tfm\_format)};
\]
\[
\text{if verbose then}
\]
\[
\begin{align*}
\text{begin} & \quad \text{print(banner)}; \quad \text{print\_ln(version\_string)}; \\
\text{end};
\end{align*}
\]
See also sections 17*, 28*, 33, 46, and 64.

This code is used in section 2*.

If an explicit filename isn’t given, we write to stdout.

\[
\text{Set initial values}\]
\[
\text{if optind + 1 = argc then}
\]
\[
\begin{align*}
\text{begin} & \quad \text{pl\_file} \leftarrow \text{stdout}; \\
\text{end}
\end{align*}
\]
\[
\text{else begin pl\_name} \leftarrow \text{extend\_filename(cmdline(optind + 1), \text{"pl"})}; \quad \text{rewrite(pl\_file, pl\_name)};
\]
\[
\text{end};
\]
Unpacked representation. The first thing TFtoPL does is read the entire tfm_file into an array of bytes, $tfm[0 \ldots (4 \times lf - 1)]$.

\begin{verbatim}
define index ≡ index_type
\langle Types in the outer block 18* \rangle ≡
  byte = 0 \ldots 255; \ \{ \text{unsigned eight-bit quantity} \}
  index = integer; \ \{ \text{address of a byte in tfm} \}
\end{verbatim}

See also section 107*.

This code is used in section 2*.

\begin{verbatim}
define abort(\#) ≡
begin print ln(\#);
  print ln(‘Sorry, \underline{I} can’t go on; \underline{are} you sure this \underline{is} a \underline{TFM}?’);
  goto final_end;
end
\end{verbatim}

\begin{verbatim}
(Read the whole input file 20*) ≡
read (tfm_file, tfm[0]);
if tfm[0] > 127 then abort(‘The first byte of the input file exceeds 127!’);
if eof(tfm_file) then abort(‘The input file is only one byte long!’);
read (tfm_file, tfm[1]); if $lf \leftarrow tfm[0] \times 400 + tfm[1]$;
if $lf = 0$ then abort(‘The file claims to have length zero, but that’s impossible!’);
$tfm_file\_array \leftarrow \text{xrealloc\_array}(tfm\_file\_array, byte, 4 \times lf + 1000)$;
for $tfm\_ptr \leftarrow 2$ to $4 \times lf - 1$ do
  begin if eof(tfm_file) then abort(‘The file has fewer bytes than it claims!’);
      read (tfm_file, tfm[tfm_ptr]);
  end;
if ¬eof(tfm_file) then
  begin print ln(‘There’s some extra junk at the end of the TFM file,’);
      print ln(‘but I’ll proceed as if it weren’t there.’);
  end
\end{verbatim}

This code is used in section 96.
In order to stick to standard Pascal, we use three strings called ASCII04, ASCII10, and ASCII14, in terms of which we can do the appropriate conversion of ASCII codes. Three other little strings are used to produce face codes like MIE.

procedure out_char(c: byte); { outputs a character }
begin
  if (font_type > vanilla) ∨ (charcode_format = charcode_octal) then
    begin if (c > \u1) ∧ (c ≤ \n) ∧ (c ≠ \( ) ∧ (c ≠ \)) then
      out(\C, ASCIIall[c - \u + 1]) { default case, use C only for letters and digits }
    else if (c ≥ \0) ∧ (c ≤ \9) then out(\C, c - \0 : 1)
    else if (c ≥ \A) ∧ (c ≤ \Z) then out(\C, ASCII10[c - \A + 2])
    else if (c ≥ \a) ∧ (c ≤ \z) then out(\C, ASCII14[c - \a + 2])
    else begin tfm[0] ← c; out_octal(0,1);
    end;
  else begin
    put_byte(ASCIIall[1 + (b mod 3)],pl_file);
    put_byte(ASCIIall[1 + (b div 3)],pl_file);
    end;
end;

The property value might be a “face” byte, which is output in the curious code mentioned earlier, provided that it is less than 18.

procedure out_face(k : index); { outputs a face }
var
  s: 0..1; { the slope }
  b: 0..8; { the weight and expansion }
begin
  if tfm[k] ≥ 18 then out_octal(k,1)
  else begin
    out(\F, ); { specify face-code format }
    s ← tfm[k] mod 2; b ← tfm[k] div 2; put_byte(ASCIIall[1 + (b mod 3)],pl_file);
    put_byte(ASCIIall[1 + s],pl_file);
    end;
end;
The last thing on TFtoPL’s agenda is to go through the list of `char_info` and spew out the information about each individual character.

\[\text{Do the characters } 78^*\] \equiv
\[
sort_ptr \leftarrow 0; \quad \{ \text{this will suppress ‘STOP’ lines in ligature comments} \}
\]
\[
\text{for } c \leftarrow bc \text{ to } ec \text{ do }
\]
\[\text{if } \text{width_index}(c) > 0 \text{ then }
\]
\[\begin{array}{ll}
\text{begin} & \text{if } \text{chars_on_line} = 8 \text{ then }
\]
\[\begin{array}{ll}
& \text{begin} \text{ print}_\text{ln}(\text{‘\_’}); \text{ chars_on_line} \leftarrow 1;
\end{array}
\]
\[\text{end}
\]
\[\text{else begin if } \text{chars_on_line} > 0 \text{ then } \text{print}(\text{‘\_’});
\]
\[\text{if } \text{verbose then } \text{incr}(\text{chars_on_line});
\]
\[\text{end};
\]
\[\text{if } \text{verbose then } \text{print}_\text{octal}(c); \quad \{ \text{progress report} \}
\]
\[\text{left}; \text{ out(‘CHARACTER’); out_char(c); out_ln}; \quad \{ \text{Output the character’s width} 79\};
\]
\[\text{if } \text{height_index}(c) > 0 \text{ then } \{ \text{Output the character’s height} 80\};
\]
\[\text{if } \text{depth_index}(c) > 0 \text{ then } \{ \text{Output the character’s depth} 81\};
\]
\[\text{if } \text{italic_index}(c) > 0 \text{ then } \{ \text{Output the italic correction} 82\};
\]
\[\text{case } \text{tag}(c) \text{ of}
\]
\[\text{no_tag: do nothing};
\]
\[\text{lig_tag: } \{ \text{Output the applicable part of the ligature/kern program as a comment} 83\};
\]
\[\text{list_tag: } \{ \text{Output the character link unless there is a problem} 84\};
\]
\[\text{ext_tag: } \{ \text{Output an extensible character recipe} 85\};
\]
\[\text{end}; \quad \{ \text{there are no other cases} \}
\]
\[\text{right};
\]

This code is used in section 98.
To detect such loops, TToPL attempts to evaluate the function \( f(x, y) \) for all character pairs \( x \) and \( y \), where \( f \) is defined as follows: If the current character is \( x \) and the next character is \( y \), we say the “cursor” is between \( x \) and \( y \); when the cursor first moves past \( y \), the character immediately to its left is \( f(x, y) \). This function is defined if and only if no infinite loop is generated when the cursor is between \( x \) and \( y \).

The function \( f(x, y) \) can be defined recursively. It turns out that all pairs \((x, y)\) belong to one of five classes. The simplest class has \( f(x, y) = y \); this happens if there’s no ligature between \( x \) and \( y \), or in the cases LIG/> and /LIG>>. Another simple class arises when there’s a LIG or /LIG> between \( x \) and \( y \), generating the character \( z \); then \( f(x, y) = z \). Otherwise we always have \( f(x, y) \) equal to either \( f(x, z) \) or \( f(z, y) \) or \( f(f(x, z), y) \), where \( z \) is the inserted ligature character.

The first two of these classes can be merged; we can also consider \((x, y)\) to belong to the simple class when \( f(x, y) \) has been evaluated. For technical reasons we allow \( x \) to be 256 (for the boundary character at the left) or 257 (in cases when an error has been detected).

For each pair \((x, y)\) having a ligature program step, we store \((x, y)\) in a hash table from which the values \( z \) and \( \text{class} \) can be read.

```plaintext
define simple = 0  { f(x, y) = z }
define left_z = 1  { f(x, y) = f(z, y) }
define right_z = 2  { f(x, y) = f(x, z) }
define both_z = 3  { f(x, y) = f(f(x, z), y) }
define pending = 4  { f(x, y) is being evaluated }
define class ≡ class_var
```

(Global in the outer block 6) +≡

\[ \text{hash: array [0..hash_size] of 0..66048; \{ 256x + y + 1 for } x \leq 257 \text{ and } y \leq 255 \} \]

\[ \text{class: array [0..hash_size] of simple..pending;} \]

\[ \text{lig_z: array [0..hash_size] of 0..257;} \]

\[ \text{hash_ptr: 0..hash_size; \{ the number of nonzero entries in hash \}} \]

\[ \text{hash_list: array [0..hash_size] of 0..hash_size; \{ list of those nonzero entries \}} \]

\[ h, hh: 0..hash_size; \{ indices into the hash table \} \]

\[ x_{lig_cycle}, y_{lig_cycle}: 0..256; \{ problematic ligature pair \} \]
CHECKING FOR LIGATURE LOOPS

\[ \text{CHECKING FOR LIGATURE LOOPS} \]

\texttt{CHECKING FOR LIGATURE LOOPS} \texttt{\textendash} \texttt{TFtoPL changes for C} \texttt{\S 90}  

\[ 90^* \quad \text{(Check for ligature cycles \texttt{90}*)} \]

\texttt{\texttt{hash_ptr} \leftarrow 0; \texttt{y_lig_cycle} \leftarrow 256;}  
\texttt{for \texttt{hh} \leftarrow 0 \texttt{to} \texttt{hash_size} \texttt{do} \texttt{hash[hh]} \leftarrow 0; \quad \{ \text{clear the hash table} \}\ }
\texttt{for \texttt{c} \leftarrow \texttt{bc} \texttt{to} \texttt{ec} \texttt{do}}
\hspace{1em} \texttt{if \texttt{tag(c)} = \texttt{lig_tag} then}
\hspace{2em} \texttt{begin}
\hspace{3em} \texttt{i} \leftarrow \texttt{remainder(c)};
\hspace{3em} \texttt{if \texttt{tfm[lig_step(i)] > stop_flag} then \texttt{i} \leftarrow 256 \times \texttt{tfm[lig_step(i)] + 2} + \texttt{tfm[lig_step(i)] + 3];}
\hspace{3em} \langle \text{Enter data for character} \texttt{c} \texttt{starting at location} \texttt{i} \texttt{in the hash table} \rangle;
\hspace{2em} \texttt{end};
\hspace{1em} \text{if \texttt{bchar_label} < \texttt{nl} then}
\hspace{2em} \texttt{begin}
\hspace{3em} \texttt{c} \leftarrow 256; \texttt{i} \leftarrow \texttt{bchar_label};
\hspace{3em} \langle \text{Enter data for character} \texttt{c} \texttt{starting at location} \texttt{i} \texttt{in the hash table} \rangle;
\hspace{2em} \texttt{end};
\hspace{1em} \text{if \texttt{hash_ptr} = \texttt{hash_size} then}
\hspace{2em} \texttt{begin}
\hspace{3em} \texttt{print} \texttt{ln}('Sorry, \texttt{I\textendash}haven\textendash't\textunderscore room\textendash for\textunderscore so\textunderscore many\textunderscore ligature/kern\textunderscore pairs!'); \texttt{uexit(1);} ;
\hspace{2em} \texttt{end};
\hspace{1em} \text{for \texttt{hh} \leftarrow 1 \texttt{to} \texttt{hash_ptr} \texttt{do}}
\hspace{2em} \texttt{begin}
\hspace{3em} \texttt{r} \leftarrow \texttt{hash_list[hh]};
\hspace{3em} \texttt{if \texttt{class[r]} > \texttt{simple} then} \quad \{ \text{make sure} \texttt{f} \texttt{is defined} \}
\hspace{4em} \texttt{r} \leftarrow \texttt{f_fn(r, (hash[r] - 1) div 256, (hash[r] - 1) mod 256)};
\hspace{2em} \texttt{end};
\hspace{1em} \text{if \texttt{y_lig_cycle} < 256 then}
\hspace{2em} \texttt{begin}
\hspace{3em} \texttt{print('Infinite ligature loop with \langle \text{X} \rangle');}
\hspace{3em} \texttt{if \texttt{x_lig_cycle} = 256 then \texttt{print('boundary, else \texttt{print_octal(x_lig_cycle)};}}
\hspace{3em} \langle \texttt{\textendash and \textendash} \rangle; \texttt{print_octal(y_lig_cycle); \texttt{print}\texttt{ln('!');}}
\hspace{3em} \texttt{out('INFINITE_LIGATURE_LOOP_MUST_BE_BROKEN!\langle \text{X} \rangle'); uexit(1);} \texttt{end}
\]

This code is used in section \texttt{66}.

\[ 94^* \quad \text{Evaluation of} \texttt{f(x, y)} \text{is handled by two mutually recursive procedures. Kind of a neat algorithm,}
\]

\texttt{generalizing a depth-first search.}

\texttt{ifdef('notdef')}
\hspace{1em} \texttt{function f_fn(h, x, y : index): index;}
\hspace{2em} \texttt{begin end;}
\hspace{3em} \{ \text{compute} \texttt{f} \texttt{for arguments known to be in} \texttt{hash[h]} \}
\texttt{endif('notdef')}
\hspace{1em} \texttt{function eval(x, y : index): index; \{ compute} \texttt{f(x, y) with hashtable lookup} \}
\hspace{2em} \texttt{var key: integer; \{ value sought in hash table \}}
\hspace{3em} \texttt{begin}
\hspace{4em} \texttt{key} \leftarrow 256 \times x + y + 1; \texttt{h} \leftarrow (1009 \times \texttt{key}) \texttt{mod hash_size;}
\hspace{4em} \texttt{while} \texttt{hash[h]} > \texttt{key} \texttt{do}
\hspace{5em} \texttt{if} \texttt{h} > 0 \texttt{then} \texttt{decr(h)} \texttt{else} \texttt{h} \leftarrow \texttt{hash_size;}
\hspace{4em} \texttt{if} \texttt{hash[h]} < \texttt{key} \texttt{then} \texttt{eval} \leftarrow \texttt{y} \quad \{ \text{not in ordered hash table} \}
\hspace{5em} \texttt{else} \texttt{eval} \leftarrow \texttt{f_fn(h, x, y);}  
\hspace{3em} \texttt{end;}

Pascal's beastly convention for forward declarations prevents us from saying function $f(h, x, y : index)$: index here.

function $f_{fn}(h, x, y : \text{index}) : \text{index}$;
begin case class[h] of
  simple: do nothing;
  left_z: begin class[h] $\leftarrow$ pending; lig_z[h] $\leftarrow$ eval(lig_z[h], y); class[h] $\leftarrow$ simple;
  end;
  right_z: begin class[h] $\leftarrow$ pending; lig_z[h] $\leftarrow$ eval(x, lig_z[h]); class[h] $\leftarrow$ simple;
  end;
  both_z: begin class[h] $\leftarrow$ pending; lig_z[h] $\leftarrow$ eval(eval(x, lig_z[h]), y); class[h] $\leftarrow$ simple;
  end;
  pending: begin x_lig_cycle $\leftarrow$ x; y_lig_cycle $\leftarrow$ y; lig_z[h] $\leftarrow$ 257; class[h] $\leftarrow$ simple;
  end;  
  { the value 257 will break all cycles, since it's not in hash }
end;  { there are no other cases }
f_{fn} $\leftarrow$ lig_z[h];
end;
The main program TFtoPL changes for C

This is where TFtoPL begins and ends.

```
begin initialize;
if ¬organize then uexit(1);
do_simple_things;
⟨ Do the ligatures and kerns 66 ⟩;
⟨ Check the extensible recipes 87 ⟩;
do_characters;
if verbose then print ln(′.′);
if level ≠ 0 then print ln(′This program isn′t working!′);
if ¬perfect then
  begin out(′(COMMENT THE TFM FILE WAS BAD, SO THE DATA HAS BEEN CHANGED!)′);
    write ln(pl_file);
  end;
end.
```
§100  System-dependent changes.  Parse a Unix-style command line.

```pascal
define argument_is(#) ≡ (strcmp(long_options[option_index].name, #) = 0)
(Define parse_arguments 100*)

procedure parse_arguments;
  const n_options = 4;  { 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;
begin  { Initialize the option variables 105*};
  (Define the option table 101*);
  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 getopt_return_val = "?" then
      begin usage(my_name);
        end
    else if argument_is(‘help’) then
      begin usage_help(TFTOPL_HELP, nil);
        end
    else if argument_is(‘version’) then
      begin print_version_and_exit(banner, nil, ‘D.E. Knuth’, nil);
        end
    else if argument_is(‘charcode-format’) then
      begin if strcmp(optarg, ‘ascii’) = 0 then charcode_format ← charcode_ascii
        else if strcmp(optarg, ‘octal’) = 0 then charcode_format ← charcode_octal
          else printf(‘Bad character code format’, stringcast(optarg), ‘.’);
        end
      until getopt_return_val = −1;  { Now optind is the index of first non-option on the command line. }
  if (optind + 1 ≠ argc) ∧ (optind + 2 ≠ argc) then
    begin printf(my_name, ‘: Need one or two file arguments.’); usage(my_name);
      end;
  tfm_name ← cmdline(optind);
  end;
```

This code is used in section 2*.

101*  Here are the options we allow.  The first is one of the standard GNU options.

```pascal
(Define the option table 101*)
  current_option ← 0; long_options[current_option].name ← ‘help’;
  long_options[current_option].has_arg ← 0; long_options[current_option].flag ← 0;
  long_options[current_option].val ← 0; incr(current_option);
```

See also sections 102*, 103*, 106*, and 111*.

This code is used in section 100*.

102*  Another of the standard options.

```pascal
(Define the option table 101*)
  long_options[current_option].name ← ‘version’; long_options[current_option].has_arg ← 0;
  long_options[current_option].flag ← 0; long_options[current_option].val ← 0; incr(current_option);
```
103* Print progress information?
⟨Define the option table 101∗⟩ ≡
   long_options[current_option].name ← "verbose";
   long_options[current_option].has_arg ← 0;
   long_options[current_option].flag ← address_of(verbose);
   long_options[current_option].val ← 1;
   incr(current_option);

104* ⟨Globals in the outer block 6⟩ ≡
verbose: c_int_type;

105* ⟨Initialize the option variables 105∗⟩ ≡
   verbose ← false;
See also section 110∗.
This code is used in section 100∗.

106* This option changes how we output character codes.
⟨Define the option table 101∗⟩ ≡
   long_options[current_option].name ← "charcode-format";
   long_options[current_option].has_arg ← 1;
   long_options[current_option].flag ← 0;
   long_options[current_option].val ← 0;
   incr(current_option);

107* We use an “enumerated” type to store the information.
⟨Types in the outer block 18∗⟩ ≡
   charcode_format_type = charcode_ascii . charcode_default;

108* ⟨Constants in the outer block 4∗⟩ ≡
   charcode_ascii = 0; charcode_octal = 1; charcode_default = 2;

109* ⟨Globals in the outer block 6⟩ ≡
charcode_format: charcode_format_type;

110* It starts off as the default, that is, we output letters and digits as ASCII characters, everything else
   in octal.
⟨Initialize the option variables 105∗⟩ ≡
   charcode_format ← charcode_default;

111* An element with all zeros always ends the list.
⟨Define the option table 101∗⟩ ≡
   long_options[current_option].name ← 0;
   long_options[current_option].has_arg ← 0;
   long_options[current_option].flag ← 0;
   long_options[current_option].val ← 0;

112* Global filenames.
⟨Globals in the outer block 6⟩ ≡
tfm_name, pl_name: const c_string;
113* 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, 2, 3, 4, 7, 17, 18, 19, 20, 27, 28, 38, 39, 78, 89, 90, 94, 95, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113.

- charcode_format: 106*
- help: 101*
- verbose: 103*
- version: 102*
 a: 36, 40.
 abort: 20* 21.
 accessible: 65, 68, 69, 70, 75.
 act: 65, 71.
 activity: 65, 66, 67, 68, 69, 70, 71, 73, 75, 98.
 address_of: 100* 103*
 ai: 65, 66, 70, 75, 98.
 arg: 17* 100*
 argument_is: 100*
 argv: 2* 100*
 ASCII_all: 27* 28* 38*
 ASCII_04: 27* 28* 35.
 ASCII_10: 27* 28* 35, 38*
 ASCII_14: 27* 28* 35, 38*
 axis_height: 15.
 b: 36, 39*
 bad: 47, 50, 52, 60, 62, 70, 74, 76, 84.
 Bad TFM file: 47.
 bad_char: 47, 84, 87.
 bad_char_tail: 47.
 bad_design: 50, 51.
 banner: 1* 7* 100*
 be: 8, 9, 11, 13, 21, 23, 24, 47, 67, 78, 90*.
 bchar_label: 63, 64, 69, 90*
 big_op_spacing1: 15.
 big_op_spacing5: 15.
 boolean: 45, 96.
 bot: 14.
 both_z: 89, 92, 93, 95*
 boundary_char: 63, 64, 69, 76, 77.
 byte: 2* 19* 19*, 20* 31, 38*, 52, 98.
 c: 38*, 47, 52, 98.
 c_int_type: 100*, 104*
 cc: 63, 68, 69, 72, 92, 93.
 char: 27*
 char_info: 11, 22, 24, 78.*
 char_info_word: 9, 11, 12.
 Character list link...: 84.
 charcode_ascii: 38* 100* 107* 108*
 charcode_default: 107*, 108* 110*
 charcode_format: 38* 100* 109* 110*
 charcode_format_type: 107* 109*
 charcode_octal: 38* 100* 108*
 chars_on_line: 45, 46, 47, 78*
 check sum: 10.
 check_BCPL: 52, 53, 55.
 check_fix: 60, 62.
 check_fix_tail: 60.
 check_sum: 24, 49, 56.
 class: 89* 90* 92, 95*
 class_var: 89*
 cmdline: 17, 100*
 coding scheme: 10.
 const_c_string: 27*, 112*
 correct_bad_char: 47, 76, 77.
 correct_bad_char_tail: 47.
 count: 47, 75.
 current_option: 100* 101*, 102*, 103*, 106*, 111*
 Cycle in a character list: 84.
 d: 47.
 decr: 5, 30, 34, 35, 37, 43, 68, 92, 94*.
 default_rule_thickness: 15.
 delim1: 15.
 delim2: 15.
 delta: 40, 42.
 denom1: 15.
 denom2: 15.
 depth: 11, 24, 81.
 Depth index for char: 81.
 Depth n is too big: 62.
 depth_base: 22, 23, 24, 62.
 depth_index: 11, 24, 78*, 81.
 design size: 10.
 Design size wrong: 50.
 design_size: 24, 51.
 DESIGNSIZE IS IN POINTS: 51.
 dig: 29, 30, 31, 36, 37, 40, 41.
 do_characters: 98, 99*
 do_nothing: 5, 78*, 93, 95*, 100*
 do_simple_things: 97, 99*
 ec: 8, 9, 11, 13, 21, 23, 24, 47, 67, 78, 90*
 endif: 94*
 eof: 20*
 eval: 94*, 95*
 eval_two_bytes: 21.
 ext_tag: 12, 78*
 exten: 12, 24, 86.
 exten_base: 22, 23, 24, 87.
 extend_filename: 17*
 Extensible index for char: 85.
 Extensible recipe involves...: 87.
extensible_recipe: 9, 14.
extra_space: 15.
f: 40* 95*.
f_fn: 90* 94* 95*.
face: 10, 27* 39*.
false: 47, 67, 69, 96, 105*.
family: 24* 55*.
family name: 10.
final_end: 3* 20* 96*.
fix_word: 9, 10, 15* 24, 40, 60, 62*.
flag: 101* 102* 103* 106* 111*.
font identifier: 10.
font_type: 25, 38* 48, 53, 59, 61*.
forward: 95*.
g getopt: 100*.
g getopt_long_only: 100*.
g getopt_return_val: 100*.
g getopt_struct: 100*.
h: 89* 94* 95*.
has_arg: 101* 102* 103* 106* 111*.
hash: 89* 90* 92, 94* 95*.
hash_input: 91, 92*.
hash_list: 89* 90* 92*.
hash_ptr: 89* 90* 92*.
hash_size: 4* 89* 90* 92, 94*.
header: 10*.
height: 11, 24, 80*.
Height index for char...: 80*.
Height n is too big: 62*.
height_base: 22, 23, 24, 62*.
height_index: 11, 24, 78*, 80*.
hh: 89* 90*.
i: 47, 97*.
ifdef: 94*.
Incomplete subfiles...: 21*.
incr: 5, 34, 35, 36, 37, 41, 68, 72, 75, 78*, 92, 101* 102*, 103*, 106*.
index: 18* 35, 36, 39* 40, 47, 52, 94*, 95*, 96, 98*.
index_type: 18*.
Infinite ligature loop...: 90*.
initialize: 2* 99*.
integer: 18* 22, 30, 40, 92, 94*, 100*.
italic: 11, 24, 82*.
Italic correction index for char...: 82*.
Italic correction n is too big: 62*.
italic_base: 22, 23, 24, 62*.
italic_index: 11, 24, 78*, 82*.

j: 31, 36, 40, 52*.
k: 35, 36, 39* 40, 47, 52, 98*.
kern: 13, 24, 62, 76*.
Kern index too large: 76*.
Kern n is too big: 62*.

Kern step for nonexistent...: 76*.
kern_base: 22, 23, 24*.
kern_flag: 13, 74, 76, 93*.
key: 92, 94*.
kps_init_prog: 2*.
kps_open_file: 7*.
kps_set_program_name: 2*.
kps_tfm_format: 7*.
l: 34, 35, 36, 52*.
label_ptr: 63, 64, 67, 68, 69*.
label_table: 63, 64, 67, 68, 69, 72*.
left: 34, 49, 51, 54, 55, 56, 57, 58, 60, 66, 69, 72, 73, 76, 77, 78*, 79, 80, 81, 82, 83, 84, 85, 86*.
left_z: 89*, 93, 95*.
level: 32, 33, 34, 71, 73, 74, 99*.
if: 8, 18*, 20*, 21*.
lh: 8, 9, 21, 23, 48, 56, 57*.
Lig...skips too far: 70*.
lig Kern: 4*, 12, 13*.
lig kern_base: 22*, 23, 24*.
lig kern_command: 9, 13*.
lig size: 4* 21, 63, 65, 67, 98*.
lig step: 24, 67, 69, 70, 74, 83, 90*, 91, 93*.
lig_tag: 12, 63, 67, 78*, 90*.
lig z: 89*, 92, 95*.
Ligature step for nonexistent...: 77*.
Ligature step produces...: 77*.
Ligature unconditional stop...: 74*.
Ligature/kern starting index...: 67, 69*.
list tag: 12, 78*, 84*.
long options: 100*, 101*, 102*, 103*, 106*, 111*.
math: 25, 53, 59, 61*.
mathsy: 25, 53, 59, 61*.
M BL_string: 27*, 28*, 39*.
mid: 14*.
my name: 1*, 2*, 100*.
n options: 100*.
name: 100*, 101*, 102*, 103*, 106*, 111*.
n d: 8, 9, 21, 23, 62, 81*.
ne: 8, 9, 21, 23, 85, 87*.
next char: 13*.
Nh: 8, 9, 21, 23, 62, 80*.
n i: 8, 9, 21, 23, 62, 82*.
n k: 8, 9, 21, 23, 62, 76*.
n l: 8, 9, 13, 21, 23, 66, 67, 69, 70, 71, 74, 83, 90*, 91*.
n o tag: 12, 24, 78*, 91*.
nonexistent: 24, 76, 77, 84, 86, 87*.
Nonstandard ASCII code...: 52*.
nonzero fix: 62*.
n p: 8, 9, 21, 58, 59*.
num1: 15*.
num2: 15*.
num3: 15.
num: 8, 9, 21, 23, 62, 79.
add: 77.
One of the subfile sizes...: 21.
opbyte: 13.
optary: 100*
option: 17* 100*
option_index: 100*
organize: 96, 99*
out: 26, 30, 34, 35, 36, 38* 39* 40, 42, 43, 49, 50,
51, 54, 55, 56, 57, 58, 60, 61, 66, 69, 72, 73, 75,
76, 77, 78* 79, 80, 81, 82, 83, 84, 85, 86, 90* 99*
out_BCP: 35, 54, 55.
out_char: 38* 69, 72, 76, 77, 78* 84, 86.
out_digs: 30, 36, 41.
out_face: 39* 56.
out_fix: 40, 51, 60, 76, 79, 80, 81, 82.
out_ln: 34, 51, 58, 66, 73, 75, 78* 83, 85.
out_octal: 36, 38*, 39*, 49, 56.
output: 2*
param: 10, 15, 24, 60.
Parameter n is too big: 60.
Parenthesis...changed to slash: 52.
parse_arguments: 2* 100*
pending: 89* 95*
perfect: 45, 46, 47, 67, 69, 99*
pl_file: 2* 16, 17* 26, 34, 39* 99*
pl_name: 17* 112*
print: 2* 7* 30, 31, 47, 67, 69, 78* 84, 90*.
print_digs: 30, 31.
print_ln: 2* 7* 20* 47, 50, 59, 60, 67, 69, 70, 77,
78* 84, 90* 99* 100*.
print_octal: 31, 47, 67, 78* 84, 90*.
print_version_and_exit: 100*.
put_byte: 39*
quad: 15.
r: 47.
random_word: 24, 56, 57.
rang_error: 47, 79, 80, 81, 82, 85.
RCE_string: 27* 28* 39*.
read: 20*.
remainder: 11, 12, 13, 24, 67, 83, 84, 85, 90*.
rep: 14.
reset_tag: 24, 67, 84, 85.
rewrite: 17*.
RL_string: 27* 28* 39*.
right: 34, 49, 51, 54, 55, 56, 57, 58, 60, 66, 69, 71,
72, 73, 76, 77, 78* 79, 80, 81, 82, 83, 84, 85, 86.
right_z: 89* 93, 95*.
rr: 63, 64, 67, 68, 69, 72.
s: 39*
scheme: 24, 53, 54.
seven_bit_safe_flag: 10, 57.
should be zero: 62.
simple: 89* 90* 92, 93, 95*.
skip_byte: 13.
slant: 15.
Sorry, I haven't room...: 90*.
sort_ptr: 63, 68, 71, 72, 78*.
space: 15.
space_shrink: 15.
space_stretch: 15.
stderr: 2*
stdout: 17*.
stop_flag: 13, 67, 70, 74, 75, 83, 90* 91.
strcat: 28*.
strcmp: 100*.
strcpy: 28*.
String is too long...: 52.
stringcast: 100*.
stuff: 9.
subdrop: 15.
Subfile sizes don't add up...: 21.
sub1: 15.
sub2: 15.
supdrop: 15.
sup1: 15.
sup2: 15.
sup3: 15.
system dependencies: 2* 38*.
t: 92.
tag: 11, 12, 24, 63, 67, 78* 84, 90*.
text: 16.
tfm: 18* 19* 20* 21, 22, 24, 35, 36, 37, 38* 39* 40,
47, 51, 52, 53, 57, 60, 62, 67, 69, 70, 74, 75,
76, 77, 83, 86, 87, 90* 91, 93, 96.
tfm_file: 2* 6, 7* 18* 20*.
tfm_file_array: 2* 19* 20*.
tfm_name: 7* 100* 112*.
tfm_ptr: 20* 21, 96.
TFToPL: 2*.
TFTOPL_HELP: 100*.
The character code range...: 21.
The file claims...: 20*.
The file has fewer bytes...: 20*.
The first byte...: 20*.
The header length...: 21.
The input...one byte long: 20*.
The lig/kern program...: 21.
THE TFM FILE WAS BAD...: 99*.
There are... recipes: 21.
There's some extra junk...: 20*.
This program isn’t working: 99*

* top: 14.
* true: 46, 96.
* uexit: 90*, 99*
* unreachable: 65, 66, 67, 73.

Unusual number of fontdimen...: 59.

* usage: 100*
* usage_help: 100*
* val: 101*, 102*, 103*, 106*, 111*
* verbose: 7*, 78*, 99*, 103*, 104*, 105*
* version_string: 7*
* width: 11, 24, 62, 79.

Width n is too big: 62.

* width_base: 22, 23, 24, 62.
* width_index: 11, 24, 78*, 79.
* write Ln: 2*, 34, 99*
* x: 94*, 95*
* x_height: 15.
* x_lig_cycle: 89*, 90*, 95*
* xmalloc_array: 2*
* xrealloc: 2*
* xrealloc_array: 20*
* y: 92, 94*, 95*
* y_lig_cycle: 89*, 90*, 95*
* zz: 92, 93.
(Build the label table 67) Used in section 66.
(Comment out unreachable steps 60) Used in section 58.
(Comment if np is complete for this font type 69) Used in section 66.
(Enter data for character c at location i in the hash table 70) Used in sections 90* and 90*.
(Globals in the outer block 71*) Used in section 2*.
(Initialize the option variables 72) Used in section 90*.
(Insert (c, r) into label table 73) Used in section 74.
(Output the applicable part of the ligature/kern program as a comment 74) Used in section 78*.
(Output the family name 75) Used in section 48.
(Output the character coding scheme 76) Used in section 48.
(Output the character link unless there is a problem 77) Used in section 78*.
(Output the character's depth 78) Used in section 78*.
(Output the character's height 79) Used in section 78*.
(Output the character's width 80) Used in section 78*.
(Output either SKIP or STOP 81) Used in section 74.
(Output the check sum 82) Used in section 48.
(Output the design size 83) Used in section 48.
(Output the extensible pieces that exist 84) Used in section 85.
(Output the family name 85) Used in section 48.
(Output the fraction part, \( f/2^{20} \), in decimal notation 86) Used in section 40.
(Output the integer part, a, in decimal notation 87) Used in section 40.
(Output the italic correction 88) Used in section 78*. 
(Output the name of parameter i 89) Used in section 60.
(Output the rest of the header 90) Used in section 48.
(Output the seven_bit_safe_flag 91) Used in section 48.
(Read the whole input file 92) Used in section 90*.
(Reduce l by one, preserving the invariants 93) Used in section 37.
(Reduce negative to positive 94) Used in section 40.
(Reduce negative to positive 95) Used in section 40.
(Reduce negative to positive 96) Used in section 40.
(Refute unreachable steps 97) Used in section 48.
(Reduce negative to positive 98) Used in section 40.
(Refute unreachable steps 99) Used in section 48.
(Reduce negative to positive 100) Used in section 40.
(Reduce negative to positive 101) Used in section 40.
(Reduce negative to positive 102) Used in section 40.
(Reduce negative to positive 103) Used in section 40.
(Reduce negative to positive 104) Used in section 40.
(Reduce negative to positive 105) Used in section 40.
(Reduce negative to positive 106) Used in section 40.
(Reduce negative to positive 107) Used in section 40.
(Reduce negative to positive 108) Used in section 40.
(Reduce negative to positive 109) Used in section 40.
(Reduce negative to positive 110) Used in section 40.
(Reduce negative to positive 111) Used in section 40.