# The GFtoDVI processor

(Version 3.0, October 1989)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>The character set</td>
<td>10</td>
</tr>
<tr>
<td>Device-independent file format</td>
<td>19</td>
</tr>
<tr>
<td>Generic font file format</td>
<td>27</td>
</tr>
<tr>
<td>Extensions to the generic format</td>
<td>34</td>
</tr>
<tr>
<td>Font metric data</td>
<td>36</td>
</tr>
<tr>
<td>Input from binary files</td>
<td>45</td>
</tr>
<tr>
<td>Reading the font information</td>
<td>52</td>
</tr>
<tr>
<td>The string pool</td>
<td>70</td>
</tr>
<tr>
<td>File names</td>
<td>86</td>
</tr>
<tr>
<td>Shipping pages out</td>
<td>102</td>
</tr>
<tr>
<td>Rudimentary typesetting</td>
<td>116</td>
</tr>
<tr>
<td>Gray fonts</td>
<td>124</td>
</tr>
<tr>
<td>Slant fonts</td>
<td>134</td>
</tr>
<tr>
<td>Representation of rectangles</td>
<td>139</td>
</tr>
<tr>
<td>Doing the labels</td>
<td>153</td>
</tr>
<tr>
<td>Doing the pixels</td>
<td>204</td>
</tr>
<tr>
<td>The main program</td>
<td>219</td>
</tr>
<tr>
<td>System-dependent changes</td>
<td>222</td>
</tr>
<tr>
<td>Index</td>
<td>232</td>
</tr>
</tbody>
</table>

The preparation of this report was supported in part by the National Science Foundation under grants IST-8201926, MCS-8300984, and CCR-8610181, and by the System Development Foundation. '\TeX' is a trademark of the American Mathematical Society. 'METAFONT' is a trademark of Addison-Wesley Publishing Company.
1* Introduction. The GFtoDVI utility program reads binary generic font ("GF") files that are produced by font compilers such as METAFONT, and converts them into device-independent ("DVI") files that can be printed to give annotated hardcopy proofs of the character shapes. The annotations are specified by the comparatively simple conventions of plain METAFONT; i.e., there are mechanisms for labeling chosen points and for superimposing horizontal or vertical rules on the enlarged character shapes.

The purpose of GFtoDVI is simply to make proof copies; it does not exhaustively test the validity of a GF file, nor do its algorithms much resemble the algorithms that are typically used to prepare font descriptions for commercial typesetting equipment. Another program, GFtype, is available for validity checking; GFtype also serves as a model of programs that convert fonts from GF format to some other coding scheme.

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

```
define my_name ≡ ´gftodvi´
define banner ≡ ´This is GFtoDVI, Version 3.0´ { printed when the program starts }
```

3* The main input and output files are not mentioned in the program header, because their external names will be determined at run time (e.g., by interpreting the command line that invokes this program). 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.

```
define print(#) ≡ write(stdout,#)
define print_ln(#) ≡ write_ln(stdout,#)
define print_nl(#) ≡ begin write_ln(stdout); write(stdout,#); end
```

4* This module deleted, since it only defined the label final_end.

8* If the GF file is badly malformed, the whole process must be aborted; GFtoDVI 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 jump_out has been introduced.

```
define abort(#) ≡ begin write_ln(stderr,#); jump_out: end
define bad_gf(#) ≡ abort(´Bad GF file: ´, #, ´! (at byte cur_loc – 1) , `)
```

```
procedure jump_out;
begin uexit(1);
end;
```
11* The original Pascal compiler was designed in the late 60s, when six-bit character sets were common, so it did not make provision for lowercase letters. Nowadays, of course, we need to deal with both capital and small letters in a convenient way. So we shall assume that the Pascal system being used for \texttt{GFtoDVI} has a character set containing at least the standard visible ASCII characters ("!" through "~"). If additional characters are present, \texttt{GFtoDVI} can be configured to work with them too.

Some Pascal compilers use the original name \texttt{char} for the data type associated with the characters in text files, while other Pascals consider \texttt{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 \texttt{text.char} to stand for the data type of the characters in the output file. We shall also assume that \texttt{text.char} consists of the elements \texttt{chr(first_text_char)} through \texttt{chr(last_text_char)}, inclusive. The following definitions should be adjusted if necessary.

\begin{verbatim}
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 }
\end{verbatim}

14* Here now is the system-dependent part of the character set. If \texttt{GFtoDVI} is being implemented on a garden-variety Pascal for which only standard ASCII codes will appear in the input and output files, you don’t need to make any changes here. But if you have, for example, an extended character set like the one in Appendix C of \texttt{The \TeX{}book}, the first line of code in this module should be changed to

\begin{verbatim}
for i ← 0 to '37 do xchr[i] ← chr(i);
\end{verbatim}

\texttt{WEB’s} character set is essentially identical to \TeX{}’s.

\begin{verbatim}
(Set initial values 13) +=
for i ← 1 to '37 do xchr[i] ← chr(i);
for i ← '177 to '377 do xchr[i] ← chr(i);
\end{verbatim}

16* The \texttt{input.in} routine waits for the user to type a line at his or her terminal; then it puts ASCII-code equivalents for the characters on that line into the \texttt{buffer} array. The \texttt{term.in} file is used for terminal input.

Since the terminal is being used for both input and output, some systems need a special routine to make sure that the user can see a prompt message before waiting for input based on that message. (Otherwise the message may just be sitting in a hidden buffer somewhere, and the user will have no idea what the program is waiting for.) We shall call a system-dependent subroutine \texttt{update_terminal} in order to avoid this problem.

\begin{verbatim}
define update_terminal ≡ fflush(stdout)  { empty the terminal output buffer }
define term_in ≡ stdin   { standard input }
\end{verbatim}

\begin{verbatim}
(Globals in the outer block 12) +=
\texttt{buffer: array [0 .. terminal_line_length] of 0 .. 255;}
\end{verbatim}

17* A global variable \texttt{line_length} records the first buffer position after the line just read.

\begin{verbatim}
procedure input.in;   { inputs a line from the terminal }
   begin update_terminal;
      if eoln(term_in) then read.in(term_in);
      line_length ← 0;
      while (line_length < terminal_line_length) ∧ ¬eoln(term_in) do
         begin buffer[line_length] ← xord[getc(term_in)]; incr(line_length);
         end;
   end;
\end{verbatim}
To prepare these files for input or output, we *reset* or *rewrite* them. An extension of Pascal is needed, 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 \( \text{`reset}(f,s)\)’ and \( \text{`rewrite}(f,s)\)’ do this, when \( f \) is a file variable and \( s \) is a string variable that specifies the file name.

In C, we do path searching based on the user’s environment or the default path. We also read the command line and print the banner here (since we don’t want to print the banner if the command line is unreasonable).

\[
\begin{align*}
\text{procedure open_gf_file;} & \quad \{ \text{prepares to read packed bytes in } \text{gf_file} \} \\
& \quad \text{begin} \quad \text{gf_file} \leftarrow \text{kpse_open_file(stringcast(name_of_file), kpse_gf_format); cur_loc} \leftarrow 0; \\
& \quad \text{end;} \\
\text{procedure open_tfm_file;} & \quad \{ \text{prepares to read packed bytes in } \text{tfm_file} \} \\
& \quad \text{begin} \quad \text{tfm_file} \leftarrow \text{kpse_open_file(stringcast(name_of_file), kpse_tfm_format);} \\
& \quad \text{end;} \\
\text{procedure open_dvi_file;} & \quad \{ \text{prepares to write packed bytes in } \text{dvi_file} \} \\
& \quad \text{begin} \quad \text{rewritebin(dvi_file, stringcast(name_of_file))}; \\
& \quad \text{end;} \\
\end{align*}
\]

If you looked carefully at the preceding code, you probably asked, “What are \( \text{cur_loc} \) and \( \text{name_of_file} \)?” Good question. They are global variables: The integer \( \text{cur_loc} \) tells which byte of the input file will be read next, and the string \( \text{name_of_file} \) will be set to the current file name before the file-opening procedures are called.

\[
\begin{align*}
\langle \text{Globals in the outer block } 12 \rangle \equiv \\
\text{cur_loc: integer; } & \quad \{ \text{current byte number in } \text{gf_file} \} \\
\text{name_of_file: \uparrow \text{text_char};}
\end{align*}
\]
Reading the font information. Now let’s get down to brass tacks and consider the more substantial routines that actually convert TFM data into a form suitable for computation. The routines in this part of the program have been borrowed from \TeX, with slight changes, since \texttt{GFtoDVI} has to do some of the things that \TeX does.

The TFM data is stored in a large array called \texttt{font_info}. Each item of \texttt{font_info} is a \texttt{memory_word}; the \texttt{fix_word} data gets converted into \texttt{scaled} entries, while everything else goes into words of type \texttt{four_quarters}. (These data structures are special cases of the more general memory words of \TeX. On some machines it is necessary to define \texttt{min_quarterword} = −128 and \texttt{max_quarterword} = 127 in order to pack four quarterwords into a single word.)

\begin{verbatim}
define min_quarterword = 0 { change this to allow efficient packing, if necessary }
define max_quarterword = 255 { ditto }
define qi(#) ≡ # + min_quarterword { to put an eight_bits item into a quarterword }
define qo(#) ≡ # − min_quarterword { to take an eight_bits item out of a quarterword }
define title_font = 1
define label_font = 2
define gray_font = 3
define slant_font = 4
define logo_font = 5
define non_char ≡ qi(256)
define non_address ≡ font_mem_size
\end{verbatim}

(\texttt{Types in the outer block 9} +≡
\begin{verbatim}
font_index = 0..font_mem_size; quarterword = min_quarterword .. max_quarterword; { 1/4 of a word }
four_quarters = \textbf{packed record} B0: quarterword;
   B1: quarterword;
   B2: quarterword;
   B3: quarterword;
end;
\end{verbatim}

\texttt{\#include"gftodmem.h";} { note the ; so \texttt{web2c} will translate types that come after this }
\begin{verbatim}
internal_font_number = title_font .. logo_font;
\end{verbatim}
Of course we want to define macros that suppress the detail of how font information is actually packed, so that we don’t have to write things like

\[
\text{font\_info}[\text{width\_base}[f] + \text{font\_info}[\text{char\_base}[f] + c].qqq.0.b0].sc
\]

too often. The WEB definitions here make \text{char\_info}(f)(c) the \text{four\_quarters} word of font information corresponding to character \(c\) of font \(f\). If \(q\) is such a word, \(\text{char\_width}(f)(q)\) will be the character’s width; hence the long formula above is at least abbreviated to

\[
\text{char\_width}(f)(\text{char\_info}(f)(c)).
\]

In practice we will try to fetch \(q\) first and look at several of its fields at the same time.

The italic correction of a character will be denoted by \(\text{char\_italic}(f)(q)\), so it is analogous to \(\text{char\_width}\). But we will get at the height and depth in a slightly different way, since we usually want to compute both height and depth if we want either one. The value of \(\text{height\_depth}(q)\) will be the 8-bit quantity

\[
b = \text{height\_index} \times 16 + \text{depth\_index},
\]

and if \(b\) is such a byte we will write \(\text{char\_height}(f)(b)\) and \(\text{char\_depth}(f)(b)\) for the height and depth of the character \(c\) for which \(q = \text{char\_info}(f)(c)\). Got that?

The tag field will be called \(\text{char\_tag}(q)\); and the remainder byte will be called \(\text{rem\_byte}(q)\).

```c
#define char_info_end(#) ≡ [#].qqqq
#define char_info(#) ≡ font_info[ char_base[#] + char_info_end
#define char_width_end(#) ≡ [#.B0 ] .sc
#define char_width(#) ≡ font_info[ width_base[#] + char_width_end
#define char_exists(#) ≡ (#.B0 > min_quarterword)
#define char_italic_end(#) ≡ (qo(#.B2)) div 4 ] .sc
#define char_italic(#) ≡ font_info[ italic_base[#] + char_italic_end
#define height_depth(#) ≡ qo(#.B1)
#define char_height_end(#) ≡ (#) div 16 ] .sc
#define char_height(#) ≡ font_info[ height_base[#] + char_height_end
#define char_depth_end(#) ≡ # mod 16 ] .sc
#define char_depth(#) ≡ font_info[ depth_base[#] + char_depth_end
#define char_tag(#) ≡ ((qo(#.B2)) mod 4)
#define skip_byte(#) ≡ qo(#.B0)
#define next_char(#) ≡ #.B1
#define op_byte(#) ≡ qo(#.B2)
#define rem_byte(#) ≡ #.B3
```
§62* Only the first two words of the header are needed by GFtoDVI.

\texttt{define store\_four\_quarters(#) \equiv}
\begin{quote}
\texttt{begin read\_tfm\_word; qw.B0 \leftarrow qi(b0); qw.B1 \leftarrow qi(b1); qw.B2 \leftarrow qi(b2);}
\texttt{qw.B3 \leftarrow qi(b3); # \leftarrow qw;}
\texttt{end}
\end{quote}

\begin{quote}
\texttt{(Read the TFM header 62*) \equiv}
\texttt{begin if lh < 2 then abend;}
\texttt{store\_four\_quarters(font\_check[f]); read\_tfm\_word;}
\texttt{if b0 > 127 then abend; \{ design size must be positive \}}
\texttt{z \leftarrow ((b0 \ast 256 + b1) \ast 256 + b2) \ast 16 + (b3 \texttt{div} 16);}
\texttt{if z < unity then abend;}
\texttt{while lh > 2 do}
\texttt{begin read\_tfm\_word; decr(lh); \{ ignore the rest of the header \}}
\texttt{end;}
\texttt{font\_dsize[f] \leftarrow z;}
\texttt{if s > 0 then z \leftarrow s;}
\texttt{font\_size[f] \leftarrow z;}
\texttt{end}
\end{quote}

This code is used in section 59.
We will also find it useful to have the following strings. (The names of default fonts will presumably be different at different sites.)

\begin{verbatim}
define gf_ext = max_keyword + 1  { string number for '.gf' }
define dvi_ext = max_keyword + 2  { string number for '.dvi' }
define tfm_ext = max_keyword + 3  { string number for '.tfm' }
define page_header = max_keyword + 4  { string number for 'Page' }
define char_header = max_keyword + 5  { string number for 'Character' }
define ext_header = max_keyword + 6  { string number for 'Ext' }
define left_quotes = max_keyword + 7  { string number for '('}
define right_quotes = max_keyword + 8  { string number for ')' }
define equals_sign = max_keyword + 9  { string number for '=' }
define plus_sign = max_keyword + 10  { string number for '+' }
define default_title_font = max_keyword + 11  { string number for the default title_font } 
define default_label_font = max_keyword + 12  { string number for the default label_font } 
define default_gray_font = max_keyword + 13  { string number for the default gray_font } 
define logo_font_name = max_keyword + 14  { string number for the font with METAFONT logo } 
define small_logo = max_keyword + 15  { string number for 'METAFONT' } 
define home_font_area = max_keyword + 16  { string number for system-dependent font area } 
\end{verbatim}

(Initialize the strings 77 + 3)

\begin{verbatim}
l ← 7; init_str7("."("2")("6")("0")("2")("g")("f")(gf_ext);
l ← 4; init_str4("."("d")("v")("i")(dvi_ext);
l ← 4; init_str7(".")("t")("f")("m")(tfm_ext);
l ← 7; init_str7("u")("P")("a")("g")("e")("u")(
page_header);
l ← 12; init_str12("u")("C")("h")("a")("x")("a")("c")("t")("e")("x")("u")
(char_header);
l ← 6; init_str6("u")("E")("x")("t")("u")
(ext_header);
l ← 4; init_str4("u")("n")("n")("n")
(left_quotes);
l ← 2; init_str2("n")("n")
(right_quotes);
l ← 3; init_str3("u")("n")("u")
equals_sign);
l ← 4; init_str4("u")("n")("n")
(plus_sign);
l ← 4; init_str4("e")("m")("x")("8")
(default_title_font);
l ← 6; init_str6("c")("m")("t")("t")("1")
("0")
(default_label_font);
l ← 4; init_str5("g")("r")("a")("y")
(default_gray_font);
l ← 5; init_str5("1")("o")("g")("o")("8")

\end{verbatim}
§81* We will be using this procedure when reading the GF file just after the preamble and just after eoc commands.

function interpret_xxx: keyword_code;
    label done, done1, not_found;
    var k: integer;  \{ number of bytes in an xxx command \}
    j: integer;  \{ number of bytes read so far \}
    l: 0 .. longest_keyword;  \{ length of keyword to check \}
    m: keyword_code;  \{ runs through the list of known keywords \}
    n1: 0 .. longest_keyword;  \{ buffered character being checked \}
    n2: pool_pointer;  \{ pool character being checked \}
    c: keyword_code;  \{ the result to return \}
    begin c ← no_operation; cur_string ← null_string;
        case cur_GF of
            no_op: goto done;
            yyy: begin k ← signed_quad; goto done;
                end;
            xxx1: k ← get_byte;
            xxx2: k ← get_two_bytes;
            xxx3: k ← get_three_bytes;
            xxx4: k ← signed_quad;
            othercases abort("internal_error");
            endcases;
        end;  \{ Read the next k characters of the GF file; change c and goto done if a keyword is recognized \}
    end;

§85* A simpler method is used for special commands between boc and eoc, since GFtoDVI doesn’t even look at them.

procedure skip_nop;
    label done;
    var k: integer;  \{ number of bytes in an xxx command \}
    j: integer;  \{ number of bytes read so far \}
    begin case cur_GF of
        no_op: goto done;
        yyy: begin k ← signed_quad; goto done;
            end;
        xxx1: k ← get_byte;
        xxx2: k ← get_two_bytes;
        xxx3: k ← get_three_bytes;
        xxx4: k ← signed_quad;
        othercases abort("internal_error");
        endcases;
        for j ← 1 to k do cur_GF ← get_byte;
    end;

88* Font metric files whose areas are not given explicitly are assumed to appear in a standard system area called \textit{home\_font\_area}. This system area name will, of course, vary from place to place. The program here sets it to `\TeXfonts'.

\begin{verbatim}
  (Initialize the strings 77) +=
l ← 0; init_str0(home_font_area);
\end{verbatim}

90* And here's the second.

\begin{verbatim}
  function more_name(c : ASCII_code): boolean;
      begin if c = "\n" then more_name ← false
              else begin if c = "/" then
                             begin area_delimiter ← pool_ptr; ext_delimiter ← 0;
                             end
                            else if c = "." then ext_delimiter ← pool_ptr;
                             str_room(1); append_char(c); \{ contribute c to the current string \}
                             more_name ← true;
                            end;
      end;
\end{verbatim}

92* Another system-dependent routine is needed to convert three strings into the \textit{name\_of\_file} value that is used to open files. The present code allows both lowercase and uppercase letters in the file name.

\begin{verbatim}
  define append_to_name(#) ≡
      begin c ← #; incr(k); name_of_file[k] ← xchr[c];
      end

  procedure pack_file_name(n, a, e : str_number);
      var k: integer; \{ number of positions filled in name_of_file \}
               c: ASCII_code; \{ character being packed \}
               j: integer; \{ index into str_pool \}
             name_length: integer;
      begin name_length ← length(a) + length(n) + length(e);
          name_of_file ← xmalloc_array(ASCII_code, name_length); k ← -1; \{ C strings start at position zero. \}
          for j ← str_start[a] to str_start[a + 1] − 1 do append_to_name(str_pool[j]);
          for j ← str_start[n] to str_start[n + 1] − 1 do append_to_name(str_pool[j]);
          for j ← str_start[e] to str_start[e + 1] − 1 do append_to_name(str_pool[j]);
          name_of_file[name_length] ← 0;
      end;
\end{verbatim}
The `start_gf` procedure obtains the name of the generic font file to be input from the command line. It opens the file, making sure that some input is present; then it opens the output file.

```pascal
procedure start_gf;
  label done;
  var arg_buffer: c_string; arg_buf_ptr: integer;
begin arg_buffer ← cmdline(optind); arg_buf_ptr ← 0;
  while (line_length < terminal_line_length) ∧ (arg_buffer[arg_buf_ptr] ≠ 0) do
    begin buffer[line_length] ← xord[ucharcast(arg_buffer[arg_buf_ptr])]; incr(line_length);
      incr(arg_buf_ptr);
    end;
  buf_ptr ← 0; buffer[line_length] ← "?";
  while buffer[buf_ptr] = " " do incr(buf_ptr);
  if buf_ptr < line_length then
    begin (Scan the file name in the buffer 95);
      if cur_ext = null_string then cur_ext ← gf_ext;
      pack_file_name(cur_name, cur_area, cur_ext); open_gf_file;
    end;
  job_name ← cur_name; pack_file_name(job_name, null_string, dvi_ext); open_dvi_file;
end;
```
The actual output of \texttt{dvi.buf}[a .. b] to \texttt{dvi_file} is performed by calling \texttt{write_dvi(a,b)}. It is safe to assume that \(a\) and \(b+1\) will both be multiples of 4 when \texttt{write_dvi(a,b)} is called; therefore it is possible on many machines to use efficient methods to pack four bytes per word and to output an array of words with one system call.

In C, we use a macro to call \texttt{fwrite} or \texttt{write} directly, writing all the bytes in one shot. Much better even than writing four bytes at a time.

To put a byte in the buffer without paying the cost of invoking a procedure each time, we use the macro \texttt{dvi_out}.

\begin{verbatim}
define dvi_out(#) ≡ begin dvi_buf[dvi_ptr] ← #; incr(dvi_ptr);
if dvi_ptr = dvi_limit then dvi_swap;
end
procedure dvi_swap; { outputs half of the buffer }
begin if dvi_ptr > (\texttt{\textasciitilde}FFFFFFFFFFF − dvi_offset) then abort('dvi\_length\_exceeds\_7FFFFFFF');
if dvi_limit = dvi_buf_size then
begin write_dvi(0, half_buf − 1); dvi_limit ← half_buf; dvi_offset ← dvi_offset + dvi_buf_size;
dvi_ptr ← 0;
end
else begin write_dvi(half_buf, dvi_buf_size − 1); dvi_limit ← dvi_buf_size;
end;
end;
\end{verbatim}

\begin{verbatim}
⟨Empty the last bytes out of dvi_buf 109⟩ ≡
if dvi_limit = half_buf then write_dvi(half_buf, dvi_buf_size − 1);
if dvi_ptr > (7FFFFFFF − dvi_offset) then abort('dvi\_length\_exceeds\_7FFFFFFF');
if dvi_ptr > 0 then write_dvi(0, dvi_ptr − 1)
\end{verbatim}

This code is used in section 115*.

Here’s a procedure that outputs a font definition.

\begin{verbatim}
define select_font(#) ≡ dvi_out(fnt_num_0 + #) { set current font to #}
procedure dvi_font_def(f : internal_font_number);
var k: integer; { index into str_pool }
begin dvi_out(fnt_def1); dvi_out(f);
dvi_out(qo(font_check[f].B0)); dvi_out(qo(font_check[f].B1)); dvi_out(qo(font_check[f].B2));
dvi_out(qo(font_check[f].B3));
dvi_four(font_size[f]); dvi_four(font_dsize[f]);
dvi_out(length(font_area[f])); dvi_out(length(font_name[f]));
⟨Output the font name whose internal number is f 112⟩;
end;
(Declare the procedure called \texttt{load_fonts} 98)\end{verbatim}
At the end of the program, we must finish things off by writing the postamble. An integer variable $k$ will be declared for use by this routine.

\[
\begin{align*}
\langle \text{Finish the DVI file and go to final_end 115*} \rangle & \equiv \\
\text{begin} & \quad \text{dvi_out(post);} \quad \{ \text{beginning of the postamble} \} \\
& \quad \text{dvi_four(last_bop); last_bop} \leftarrow \text{dvi_offset} + \text{dvi_ptr} - 5; \quad \{ \text{post location} \} \\
& \quad \text{dvi_four(25400000); dvi_four(473628672);} \quad \{ \text{conversion ratio for sp} \} \\
& \quad \text{dvi_four(1000);} \quad \{ \text{magnification factor} \} \\
& \quad \text{dvi_four(max_v); dvi_four(max_h);} \\
& \quad \text{dvi_out(0); dvi_out(3);} \quad \{ \text{`max_push' is said to be 3} \} \\
& \quad \text{dvi_out(total_pages div 256); dvi_out(total_pages mod 256);} \\
& \quad \text{if \ ~\text{ fonts_not_loaded \ then} } \\
& \quad \quad \text{for } k \leftarrow \text{title_font to logo_font \ do} \\
& \quad \quad \quad \text{if length(font_name[k])} > 0 \text{ then dvi_font_def(k);} \\
& \quad \quad \quad \text{dvi_out(post_post); dvi_four(last_bop); dvi_out(dvi_id_byte);} \\
& \quad \quad \quad k \leftarrow 4 + ((\text{dvi_buf_size} - \text{dvi_ptr}) \text{ mod } 4); \quad \{ \text{the number of 223’s} \} \\
& \quad \quad \text{while } k > 0 \text{ do} \\
& \quad \quad \quad \text{begin dvi_out(223); decr(k);} \\
& \quad \quad \quad \text{end; } \\
& \quad \quad \langle \text{Empty the last bytes out of dvi_buf 109*} \rangle; \\
& \quad \text{if verbose then print_ln(`\backarrow');} \\
& \quad \text{uexit(0);} \\
& \quad \text{end}
\end{align*}
\]
118* (Set initial values 13) \[=\]
\[
dummy\_info.B0 \leftarrow q_i(0); \quad dummy\_info.B1 \leftarrow q_i(0); \quad dummy\_info.B2 \leftarrow q_i(0); \quad dummy\_info.B3 \leftarrow q_i(0);
\]
The following error message is given when an absent slant has been requested.

```plaintext
procedure slant_complaint(r : real);
  begin if fabs(r - slant_reported) > 0.001 then
    begin print_nl(´Sorry, I can't make diagonal rules of slant´);
      print_real(r, 10, 5);
      print(´!´);
      slant_reported ← r;
    end;
  end;
```

§124   GF to DVI changes for C    GRAY         FONTS         315
The process of ferreting everything away comes to an abrupt halt when a boc command is sensed. The following steps are performed at such times:

(Process a character 164*)

\begin{align*}
& \text{begin checkfonts; (Finish reading the parameters of the boc 165);} \\
& \text{(Get ready to convert METAFONT coordinates to DVI coordinates 170*);} \\
& \text{(Output the bop and the title line 172);} \\
& \text{if verbose then} \\
& \quad \text{begin print(’ \{ total_pages : 1); update_terminal; \{ print a progress report \}} \\
& \quad \text{end;} \\
& \text{(Output all rules for the current character 173);} \\
& \text{(Output all labels for the current character 181);} \\
& \text{do_pixels; dvi_out(eop); \{ finish the page \}} \\
& \text{end}
\end{align*}

This code is used in section 219*.

170* \text{(Get ready to convert METAFONT coordinates to DVI coordinates 170*)} \equiv

\begin{align*}
& \text{if } \text{pre_min}_x < \text{min}_x * \text{unity} \text{ then } \text{offset}_x \leftarrow \text{offset}_x + \text{min}_x * \text{unity} - \text{pre_min}_x; \\
& \text{if } \text{pre_max}_y > \text{max}_y * \text{unity} \text{ then } \text{offset}_y \leftarrow \text{offset}_y + \text{max}_y * \text{unity} - \text{pre_max}_y; \\
& \text{if } \text{pre_max}_x > \text{max}_x * \text{unity} \text{ then } \text{pre_max}_x \leftarrow \text{pre_max}_x \text{ div unity} \\
& \quad \text{else } \text{pre_max}_x \leftarrow \text{max}_x; \\
& \text{if } \text{pre_min}_y < \text{min}_y * \text{unity} \text{ then } \text{pre_min}_y \leftarrow \text{pre_min}_y \text{ div unity} \\
& \quad \text{else } \text{pre_min}_y \leftarrow \text{min}_y; \\
& \text{delta}_y \leftarrow \text{round(unscl_ratio} * (\text{max}_y + 1) - \text{y_ratio} * \text{offset}_y) + 3276800; \\
& \text{delta}_x \leftarrow \text{round(x_ratio} * \text{offset}_x - \text{unscl_ratio} * \text{min}_x); \\
& \text{if } \slant_{\text{ratio}} \geq 0 \text{ then over_col} \leftarrow \text{round(unscl_ratio} * \text{pre_max}_x + \text{unsc_slant_ratio} \text{* max}_y) \\
& \quad \text{else over_col} \leftarrow \text{round(unscl_ratio} * \text{pre_max}_x \text{ + unsc_slant_ratio} * \text{min}_y); \\
& \text{over_col} \leftarrow \text{over_col} + \text{delta}_x + \text{overflow_label_offset}; \\
& \text{page_height} \leftarrow \text{round(unscl_ratio} * (\text{max}_y + 1 - \text{pre_min}_y)) + 3276800 - \text{offset}_y; \\
& \text{if } \text{page_height} > \text{max}_v \text{ then } \text{max}_v \leftarrow \text{page_height}; \\
& \text{page_width} \leftarrow \text{over_col} - 10000000
\end{align*}

This code is used in section 164*. 
215* define do_skip \equiv z \leftarrow 0; \text{paint\_black} \leftarrow \text{false}

define end_with(#) \equiv
  \begin{cases}#
  \end{cases}; cur\_gf \leftarrow \text{get\_byte}; \text{goto} done1; \text{end}

define five_cases(#) \equiv #, # + 1, # + 2, # + 3, # + 4

define eight_cases(#) \equiv #, # + 1, # + 2, # + 3, # + 4, # + 5, # + 6, # + 7

define thirty_two_cases(#) \equiv eight_cases(#), eight_cases(# + 8), eight_cases(# + 16), eight_cases(# + 24)

define sixty_four_cases(#) \equiv thirty_two_cases(#), thirty_two_cases(# + 32)

\langle \text{Read and process GF commands until coming to the end of this row} 215\rangle \equiv
\text{loop begin continue: if } (\text{cur\_gf} \geq \text{new\_row\_0}) \land (\text{cur\_gf} \leq \text{new\_row\_0} + 164) \text{ then}
  \end{cases}(z \leftarrow \text{cur\_gf} - \text{new\_row\_0}; \text{paint\_black} \leftarrow \text{true})
\text{else case } \text{cur\_gf} \text{ of}
  \begin{cases}
    \text{sixty\_four\_cases}(0); k \leftarrow \text{cur\_gf};
    \text{paint1}: k \leftarrow \text{get\_byte};
    \text{paint2}: k \leftarrow \text{get\_two\_bytes};
    \text{paint3}: k \leftarrow \text{get\_three\_bytes};
    \text{eoc}: \text{goto} done1;
    \text{skip0}: \text{end\_with}(\text{blank\_rows} \leftarrow 0; \text{do\_skip});
    \text{skip1}: \text{end\_with}(\text{blank\_rows} \leftarrow \text{get\_byte}; \text{do\_skip});
    \text{skip2}: \text{end\_with}(\text{blank\_rows} \leftarrow \text{get\_two\_bytes}; \text{do\_skip});
    \text{skip3}: \text{end\_with}(\text{blank\_rows} \leftarrow \text{get\_three\_bytes}; \text{do\_skip});
    \text{xxx1, xxx2, xxx3, xxx4, yyy, no\_op}: \text{begin}\text{ skip\_nop}; \text{goto} continue;
  \end{cases}\text{end};
\text{othercases } \text{bad\_gf} (\text{\textquote{Improper\_opcode}})
\text{endcases};
\langle \text{Paint } k \text{ bits and read another command} 216\rangle;
\text{end};
done1:

This code is used in section 214.
219* The main program. Now we are ready to put it all together. This is where GFtoDVI starts, and where it ends.

\begin{verbatim}
begin initialize; \{ get all variables initialized \}
   \langle Initialize the strings 77 \rangle;
start_gf; \{ open the input and output files \}
   \langle Process the preamble 221 \rangle;
cur_gf ← get_byte; init_str_ptr ← str_ptr;
loop begin \langle Initialize variables for the next character 144 \rangle;
   while (cur_gf ≥ xxx1) \&\& (cur_gf ≤ no_op) do \langle Process a no-op command 154 \rangle;
   if cur_gf = post then \langle Finish the DVI file and goto final_end 115* \rangle;
   if cur_gf ≠ boc then
      if cur_gf ≠ boc1 then \textbf{abort}('Missing \texttt{boc}!');
      \langle Process a character 164* \rangle;
   cur_gf ← get_byte; str_ptr ← init_str_ptr; pool_ptr ← str_start[str_ptr];
end;
if verbose \&\& (total_pages mod 13 ≠ 0) then print_in(\texttt{\_\_});
end.
\end{verbatim}
222* System-dependent changes. Parse a Unix-style command line.

\[\text{define } \text{argument\_is}(\#) \equiv (\text{strcmp}(\text{long\_options}[\text{option\_index}].\text{name}, \#) = 0)\]

(Define parse\_arguments 222*) \equiv

\text{procedure parse\_arguments};
\text{const } n\_options = 4; \{ \text{Pascal won't count array lengths for us.} \}
\text{var } long\_options: \text{array}[0..n\_options]\text{ of } \text{getopt}\_\text{struct};
\text{getopt\_return\_val: integer; option\_index: c\_int\_type; current\_option: 0..n\_options;}
\text{begin } \langle \text{Initialize the option variables 227*} \rangle;
\text{begin } \langle \text{Define the option table 223*} \rangle;
\text{repeat } \text{getopt\_return\_val} \leftarrow \text{getopt\_long\_only}(\text{argc}, \text{argv}, ``, long\_options, \text{address of (option\_index)});\]
\text{if } \text{getopt\_return\_val} = -1 \text{ then}
\text{begin } \text{do\_nothing; } \{ \text{End of arguments; we exit the loop below.} \}
\text{end}
\text{else if } \text{getopt\_return\_val} = `?` \text{ then}
\text{begin } \text{usage(my\_name)};\]
\text{end}
\text{else if } \text{argument\_is(`help`)} \text{ then}
\text{begin } \text{usage\_help(GFTODVI\_HELP, nil)};\]
\text{end}
\text{else if } \text{argument\_is(`version`)} \text{ then}
\text{begin } \text{print\_version\_and\_exit(banner, nil, `D.E.\_Knuth`, nil)};\]
\text{end}
\text{else if } \text{argument\_is(`overflow\_label\_offset`)} \text{ then}
\text{begin } \text{offset\_in\_points} \leftarrow \text{atof(optarg)};
\text{overflow\_label\_offset} \leftarrow \text{round}(\text{offset\_in\_points} * 65536);\]
\text{end}; \{ \text{Else it was a flag; getopt has already done the assignment.} \}
\text{until } \text{getopt\_return\_val} = -1; \{ \text{Now optind is the index of first non-option on the command line. We must have one remaining argument.} \}
\text{if } (\text{optind} + 1 \neq \text{argc}) \text{ then}
\text{begin } \text{write\_ln(stderr, my\_name, `: Need\_exactly\_one\_file\_argument. `); usage(my\_name)};\]
\text{end};\]
\text{end;}
This code is used in section 3*.

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

(Define the option table 223*) \equiv
\text{current\_option} \leftarrow 0; \text{long\_options[current\_option].name} \leftarrow `\text{help}`;
\text{long\_options[current\_option].has\_arg} \leftarrow 0; \text{long\_options[current\_option].flag} \leftarrow 0;
\text{long\_options[current\_option].val} \leftarrow 0; \text{incr(current\_option)};
See also sections 224*, 225*, 228*, and 231*.
This code is used in section 222*.

224* Another of the standard options.

(Define the option table 223*) +\equiv
\text{long\_options[current\_option].name} \leftarrow `\text{version}`; \text{long\_options[current\_option].has\_arg} \leftarrow 0;
\text{long\_options[current\_option].flag} \leftarrow 0; \text{long\_options[current\_option].val} \leftarrow 0; \text{incr(current\_option)};
\[225^*\] Print progress information?
\(\langle \text{Define the option table } 223^* \rangle \equiv \)
\[
\text{long_options}[\text{current_option}].\text{name} \leftarrow \text{"verbose"};
\text{long_options}[\text{current_option}].\text{has_arg} \leftarrow 0;
\text{long_options}[\text{current_option}].\text{flag} \leftarrow \text{address of (verbose)};
\text{long_options}[\text{current_option}].\text{val} \leftarrow 1;
\text{incr(current_option);}\]

\[226^*\] (Globals in the outer block 12) \(\equiv \)
\(\text{verbose}: \text{c_int_type};\)

\[227^*\] (Initialize the option variables 227*) \(\equiv \)
\(\text{verbose} \leftarrow \text{false};\)

This code is used in section 222*.

\[228^*\] Change how far from the right edge of the character boxes we print overflow labels.
\(\langle \text{Define the option table } 223^* \rangle \equiv \)
\[
\text{long_options}[\text{current_option}].\text{name} \leftarrow \text{"overflow-label-offset"};
\text{long_options}[\text{current_option}].\text{has_arg} \leftarrow 1;
\text{long_options}[\text{current_option}].\text{flag} \leftarrow 0;
\text{long_options}[\text{current_option}].\text{val} \leftarrow 0;
\text{incr(current_option);}\]

\[229^*\] It’s easier on the user to specify the value in \TeX{} points, but we want to store it in scaled points.
\(\langle \text{Globals in the outer block } 12 \rangle \equiv \)
\(\text{overflow_label_offset}: \text{integer}; \{ \text{in scaled points} \}\)
\(\text{offset_in_points}: \text{real};\)

\[230^*\] The default offset is ten million scaled points—a little more than two inches.
\(\langle \text{Initialize the option variables } 227^* \rangle \equiv \)
\(\text{overflow_label_offset} \leftarrow 10000000;\)

\[231^*\] An element with all zeros always ends the list.
\(\langle \text{Define the option table } 223^* \rangle \equiv \)
\[
\text{long_options}[\text{current_option}].\text{name} \leftarrow 0;
\text{long_options}[\text{current_option}].\text{has_arg} \leftarrow 0;
\text{long_options}[\text{current_option}].\text{flag} \leftarrow 0;
\text{long_options}[\text{current_option}].\text{val} \leftarrow 0;\)
232* Index. Here is a list of the section numbers where each identifier is used. Cross references to error messages and a few other tidbits of information also appear.

The following sections were changed by the change file: 1, 3, 4, 8, 11, 14, 16, 17, 47, 48, 52, 55, 62, 78, 81, 85, 88, 90, 92, 94, 107, 108, 109, 111, 115, 118, 138, 164, 170, 215, 219, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232.

- help: 223*
- overflow-label-offset: 228*
- version: 224*

a: 51, 92* 211.
abend: 58, 60, 62* 63, 64, 66.
abort: 8* 58, 61, 73, 74, 75, 81* 85* 91, 108* 109*
141, 165, 169, 184, 219*.
abs: 151, 152, 173, 178.
address_of: 222* 225*.
adjust: 69.
alpha: 58, 64, 65.
append_char: 73, 75, 83, 90* 101, 221.
append_to_name: 92*.
area_code: 77, 98, 100, 101, 154.
area_delimiter: 87, 89, 90* 91.
argv: 94*
arg_buff_ptr: 94*.
arg_buffer: 94*.
argv: 222*.
argument_is: 222*.
argv: 3* 222*.
ASCII_code: 10, 11* 12, 71, 73, 90* 92* 116.
at size: 39.
_at_code: 77, 154.
atof: 223*.
b: 51, 207.
backpointers: 32.
Bad GF file: 8*.
Bad label type...: 163.
Bad TFM file...: 58.
bad_gf: 8* 215* 221.
bad_tfm: 58.
banner: 1* 3* 222*.
bc: 37, 38, 40, 42, 58, 60, 61, 66, 69.
bch_label: 58, 66, 69.
bch_label: 53, 69, 120.
begin_name: 86, 89, 95.
bset_q: 150, 151, 152.
beta: 58, 64, 65.
BigEndian order: 19, 37.
black: 28, 29.
boc: 27, 29, 30, 31, 32, 35, 85* 96, 153, 154, 164* 165, 219*.
boc1: 29, 30, 165, 219*.
boolean: 90* 96, 116, 117, 145, 149, 194, 218, 220.
bop: 19, 21, 22, 24, 25, 102, 172.
bot: 43.

bot_coords: 186, 190, 196, 197, 198, 199.
box_depth: 116, 117, 121, 185, 186.
box_height: 116, 117, 121, 185, 186.
box_width: 116, 117, 119, 121, 185, 186.
buf_ptr: 18, 94* 95, 100, 101.
buffer: 16* 17, 18, 75, 82, 83, 94* 95, 100, 101, 114.
byte_file: 45, 46.
b0: 49, 50, 53, 55* 60, 62* 63, 64, 66, 67, 68.
b1: 52* 55* 62* 111* 118*.
B0: 52* 55* 62* 111* 118*.
b1: 49, 50, 53, 55* 56, 62* 63, 64, 66, 67, 68.
B1: 52* 55* 62* 111* 118*.
b2: 49, 50, 53, 55* 60, 62* 63, 64, 66, 67, 68.
B2: 52* 55* 62* 111* 118*.
b3: 49, 50, 60, 62* 63, 64, 66, 67, 68.
B3: 52* 55* 62* 111* 118*.
c: 51, 75, 81* 90* 92* 113, 127.
c_int_type: 222* 226*.
c_string: 94*.
char: 11*.
char_code: 165, 166, 172.
char_depth: 55* 121.
char_depth_end: 55*.
char_exists: 55* 121, 169, 184, 205.
char_header: 78* 172.
char_height: 55* 121, 137, 169, 184.
char_height_end: 55*.
char_info_end: 55*.
char_info_word: 38, 40, 41.
char_italic: 55*.
char_italic_end: 55*.
char_kern: 56, 120.
char_kern_end: 56.
char_loc: 29, 32.
char_loc0: 29.
char_width: 55* 121, 169, 184.
char_width_end: 55*.
Character too wide: 165.
check sum: 24, 31, 39.
check_byte_range: 66, 67.
check_fonts: 96, 164*.
Chinese characters: 32.
chr: 11* 12, 14* 15.
cmdline: 94*.
coding scheme: 39.
cs:  31.
cur_area:  86, 91, 94* 
cur_ext:  86, 91, 94* 
cur_gf:  79, 80, 81*82, 84, 85*154, 165, 214, 215*216, 217, 218, 219*.
cur_l:  116, 120, 121, 122, 123.
cur_loc:  8*47*48*51, 154, 163.
cur_name:  86, 91, 94* 
cur_r:  116, 120, 122, 123.
cur_string:  79, 80, 81*83, 154, 162, 163.
current_option:  222*223*224*225*228*231* 
d:  51, 127, 150.
d_min:  150, 151, 152.
decr:  2, 62*69, 75, 95, 114, 115*122, 179, 180, 210, 213, 214, 217.
default fonts:  78*. 
default_gray_font:  78*97. 
default_label_font:  78*97. 
default_rule_thickness:  44, 57, 135, 175. 
default_title_font:  78*97. 
del_m:  29. 
del_n:  29. 
delta:  183, 184, 185, 186. 
delta_x:  167, 168, 170*209, 218. 
delta_y:  167, 168, 170*218. 
den:  21, 23, 25. 
depth_base:  53, 55*61, 64. 
depth_index:  40, 55* 
design size:  31, 39. 
dfl:  194, 195, 196, 197, 198, 199. 
dm:  29. 
do_nothing:  7, 63, 154, 222* 
do_pixels:  164*218. 
do_skip:  215* 
done1:  6, 81*82, 215*218. 
dot_height:  148, 183, 184, 185, 186, 188. 
dot_width:  148, 183, 184, 186, 188. 
down1:  21. 
down2:  21. 
down3:  21. 
down4:  21, 22, 171. 
ds:  31. 
dummy_info:  117, 118*120. 
DVI files:  19. 
dvi length exceeds...:  108*109* 
dvi_buf:  104, 105, 107*108* 
dvi_buf_size:  5, 104, 105, 106, 108*109*115* 
dvi_ext:  78*94* 
dvi_file:  46, 47*107*. 
dvi_font_def:  98, 111*115*. 
dvi_four:  110, 111*115*119, 121, 171, 172, 176, 177, 179, 180, 209, 221. 
dvi_goto:  171, 172, 176, 177, 178, 188, 190, 194, 202, 218. 
dvi_id_byte:  23, 115*221. 
dvi_index:  104, 105. 
dvi_out:  108*110, 111*112, 113, 114, 115*119, 121, 164*171, 172, 176, 177, 178, 179, 180, 188, 190, 194, 202, 208, 209, 218, 221. 
dvi_swap:  108*. 
dx:  29, 32, 192, 220. 
dy:  29, 32, 179, 180, 192, 220. 
d0:  148, 150, 151, 152. 
e:  92* 
ec:  37, 38, 40, 42, 58, 60, 61, 66, 69. 
eight_bits:  45, 49, 51, 52*53, 80, 105, 113, 116, 218. 
eight_cases:  215* 
eighth_octant:  192, 196, 197, 198, 199. 
else:  2. 
end:  2. 
end_k:  116, 122, 123. 
end_name:  86, 91, 95. 
end_with:  215*. 
endcases:  2. 
eof:  51. 
eoln:  17* 
eop:  19, 21, 22, 24, 164*. 
exit:  6, 7, 145, 194, 218. 
ext:  165, 166, 172. 
ext_delimiter:  87, 89, 90*91. 
ext_header:  78*172. 
ext_tag:  41, 63. 
exten:  41. 
extensible_recipe:  38, 43. 
extra_space:  44. 
f:  58, 98, 111*116. 
fabs:  138*. 

322  INDEX GF to DVI changes for C  §232
false: \texttt{90*97, 98, 116, 120, 145, 150, 190, 194, 195, 215*218, 221, 227*}
flush: \texttt{16*}
fifth_octant: \texttt{192, 196, 197, 198, 199}.
file_name_size: \texttt{5}.
final_end: \texttt{4*}
finishing_col: \texttt{208, 216, 217, 218}.
first_dot: \texttt{148, 149, 161, 187, 201}.
first_octant: \texttt{192, 196, 197, 198, 199}.
first_string: \texttt{75, 76}.
first_text_char: \texttt{11*15}.
five_cases: \texttt{215*}
fix_word: \texttt{38, 39, 44, 52*64}.
flag: \texttt{223*224*225*228*231*}
fmem_ptr: \texttt{53, 54, 61, 63, 69}.
fnt_def1: \texttt{21, 22, 111*}
fnt_def2: \texttt{21}.

\texttt{fnt_def3: 21}.

\texttt{fnt_def4: 21}.

\texttt{fnt_num_0: 21, 22, 111*}

\texttt{fnt_num_1: 21}.

\texttt{fnt_num_63: 21}.
\texttt{fn1: 21}.

\texttt{fn2: 21}.

\texttt{fn3: 21}.

\texttt{fn4: 21}.

font_area: \texttt{96, 97, 98, 101, 111*112, 154}.
\texttt{font_at: 96, 97, 98, 101, 154}.

\texttt{font_bc: 53, 69, 120, 205}.

\texttt{font_bchar: 53, 69, 116}.

\texttt{font_change: 154}.

\texttt{font_check: 53, 62*111*}

\texttt{font_dsize: 53, 62*111*}

\texttt{font_cc: 53, 69, 120, 137, 205}.

\texttt{font_index: 52*53, 58, 116}.

\texttt{font_info: 52*53, 55*56, 57, 58, 61, 63, 64, 66, 67, 68, 120}.

\texttt{font_mem_size: 5, 52*61}.

\texttt{font_name: 96, 97, 98, 101, 111*112, 115*137, 154}.

\texttt{font_size: 53, 62*111*}

\texttt{fonts_not_loaded: 96, 97, 98, 115*154}.
found: \texttt{6, 98, 99, 100, 194, 196, 197, 198, 199}.

\texttt{four_quarters: 52*53, 55*58, 98, 116, 117, 218}.

\texttt{fourth_octant: 192, 196, 197, 198, 199}.

Fuchs, David Raymond: \texttt{19, 26, 33}.

\texttt{fudge_factor: 168, 169, 202}.

\texttt{fwrite: 107*}

\texttt{get_avail: 141, 159, 162, 163, 188}.

\texttt{get_byte: 51, 81*82, 83, 84, 85*165, 215*216, 218, 219*221}.

\texttt{get_three_bytes: 51, 81*85*215*}

\texttt{get_two_bytes: 51, 81*85*215*}

\texttt{get_yyy: 84, 154, 157, 159, 163}.
getc: \texttt{17*}
getopt: \texttt{222*}
getopt_long_only: \texttt{222*}
getopt_return_val: \texttt{222*}
getopt_struct: \texttt{222*}

\texttt{gf_ext: 78*94*}
\texttt{gf_file: 46, 47*48*51, 80}.
\texttt{gf_id_byte: 29, 221}.

\texttt{GF_to_DVI: 2}.

\texttt{GFTODVI_HELP: 222*}

\texttt{gray_fonts: 35, 39, 124}.

\texttt{gray_font: 52*58, 77, 78*97, 154, 169, 175, 181, 184, 205, 210, 218}.

\texttt{gray_rule_thickness: 173, 174, 175}.

\texttt{half_buf: 104, 105, 106, 108*109*}

\texttt{half_x_height: 183, 184, 186}.

\texttt{has_arg: 223*224*225*228*231*}

\texttt{hbbox: 116, 117, 172, 183, 185, 190, 194, 202}.

\texttt{hd: 116, 121}.

\texttt{header: 39}.

\texttt{height_base: 53, 55*61, 64}.

\texttt{height_depth: 55*121, 137, 169, 184}.

\texttt{height_index: 40, 55*}

\texttt{home_font_area: 78*88*98}.

\texttt{hppp: 31}.

\texttt{i: 3*23, 98, 116, 218}.

\texttt{incr: 7, 17*51, 73, 74, 75, 82, 83, 91, 92*94*95, 100, 101, 108*114, 116, 119, 122, 123, 128, 129, 141, 172, 180, 192, 202, 208, 217, 221, 223*224*225*228*229*}.

\texttt{info: 139, 140, 148, 162, 163, 172, 188, 190, 194, 202}.

\texttt{init_str_ptr: 71, 101, 154, 219*}

\texttt{init_str0: 75, 77, 88*}

\texttt{init_str1: 75}.

\texttt{init_str10: 75, 77}.

\texttt{init_str11: 75, 77}.

\texttt{init_str12: 75, 77, 78*}

\texttt{init_str13: 75, 77}.

\texttt{init_str2: 75, 78*}

\texttt{init_str3: 75, 78*}

\texttt{init_str4: 75, 77, 78*}

\texttt{init_str5: 75, 77, 78*}

\texttt{init_str6: 75, 77, 78*}

\texttt{init_str7: 75, 77, 78*}

\texttt{init_str8: 75, 77, 78*}

\texttt{init_str9: 75, 77}.

\texttt{initialize: 3*219*}

\texttt{input_len: 16*17*18, 99}.

\texttt{integer: 3*9, 45, 48*51, 53, 58, 75, 76, 81*85, 92*94*98, 102, 105, 110, 111*114, 134, 166}.
INDEX GF to DVI changes for C

loop: 6, 7.
m: 3* 81* 98, 114, 220.
mag: 21, 23, 24, 25.
make_string: 74, 83, 91, 101, 221.
max_depth: 140, 143, 144, 147.
max_h: 102, 103, 115* 203.
max_height: 140, 143, 144, 146.
max_k: 116.
max_keyword: 77, 78* 79, 83.
max_labels: 5, 139, 140, 141, 142, 161.
max_m: 29, 31.
max_n: 29, 31.
max_node: 140, 141, 144, 187.
max_quarterword: 52*.
max_strings: 5, 70, 74, 91.
max_v: 102, 103, 115* 170*.
max_x: 165, 166, 170* 218.
max_y: 165, 166, 167, 170* 218.
memory_word: 52* 53.
mid: 43.
min_m: 29, 31.
min_n: 29, 31.
min_quarterword: 52* 53, 55* 61, 69, 116.
min_x: 165, 166, 167, 170* 218.
min_y: 165, 166, 167*.
Missing boc: 219*.
Missing dot char: 184.
Missing pixel char: 169.
more_name: 86, 90* 95.
my_name: 1* 3* 222*.
n: 3* 92* 114.
n_options: 222*.
name: 222* 223* 224* 225* 228* 231*.
name_length: 92*.
name_of_file: 47* 48* 92*.
nb: 37, 38, 58, 60, 61, 63.
nb: 37, 38, 58, 60, 61, 63.
ne: 37, 38, 58, 60, 61, 63.
new_row_0: 29, 30, 215*.
new_row_1: 29.
new_row_164: 29.
next_char: 42, 55* 120.
nh: 37, 38, 58, 60, 61, 63.
ni: 37, 38, 58, 60, 61, 63.
nil: 7.
nk: 37, 38, 58, 60, 61, 66.
nl: 37, 38, 42, 58, 60, 61, 63, 66, 69.
No preamble: 221.
No room for TFM file: 61.
no_op: 29, 30, 32, 79, 81*85, 154, 215*219*
no_operation: 79, 81*, 154.
node_tag: 41, 63.
nodex: 143, 188, 190, 194.
nodex_pointer: 139, 140, 141, 143, 145, 149, 150, 158, 160, 185, 186, 194.
non_address: 52*53, 69, 120.
non_char: 52*53, 116, 122.
not: 19, 21, 24, 25.
not_found: 6, 81*82, 98, 99.
nump: 37, 38, 39, 60, 61, 68.
num: 21, 28, 25.
nw: 37, 38, 39, 60, 61, 63.
n1: 81*, 83, 98, 100.
n2: 81*, 83, 98, 100.
act: 194, 195, 196, 197, 198, 199.
actant: 191, 192, 194, 195.
offset_code: 77, 154.
offset_in_points: 222*229*.
offset_x: 155, 156, 157, 170*.
offset_y: 155, 156, 157, 170*.
op_byte: 42, 55*, 56, 120, 122.
open_dvi_file: 47*, 94*.
open_gf_file: 47*, 94*.
open_tfm_file: 47*, 98*.
optary: 222*.
optind: 94*, 222*.
option_index: 222*.
ord: 12.
oriental characters: 32.
otherscases: 2.
others: 2.
output: 3*,
overflow_label_offset: 170*, 222*229*230*.
overflow_line: 181, 182, 202, 203.
overlap: 145, 146, 147, 196, 197, 198, 199.
p: 143, 145, 150, 185, 186, 194, 220.
pack_file_name: 92*, 94*, 98*.
page_header: 78*, 172*.
page_height: 168, 170*.
page_width: 168, 170*, 203*.
paint_black: 215, 216, 218*.
paint_switch: 28, 29.
paint_0: 29, 30.
paint1: 29, 30, 215*.
paint2: 29, 30, 215*.
paint3: 29, 30, 215*.
param: 39, 44, 57*.
param_base: 53, 57, 61, 67, 68, 69*.
param_end: 57*.
parse_arguments: 3*222*.
place_label: 193, 194, 195*.
plus_sign: 78*, 202*.
pool_pointer: 70, 71, 81*, 87, 98, 116*.
pool_ptr: 71, 73, 74, 75, 77, 90*219*.
pool_size: 5, 70, 73*.
post: 20, 21, 22, 25, 171, 172, 176, 177, 178, 188, 190, 194, 202, 208, 218.
pop_stack: 122, 123*.
post_post: 19, 21, 22, 25, 26, 27, 29, 31, 33, 115*, 219*.
print: 23, 25, 26, 29, 31, 33, 115*.
print_lineno: 3*, 115*, 164*, 219*.
print_nln: 3*, 58, 99, 138*, 154, 163*.
print_real: 138*.
print_version_and_exit: 222*.
proofing: 32.
push: 20, 21, 22, 25, 171, 208.
put_rule: 21, 22, 176, 177.
pull1: 21.
pull2: 21.
pull3: 21.
pull4: 21.
quad: 143, 145, 220*.
quarterword: 52*, 117*.
qw: 58, 62*.
rv: 138*, 143, 145, 220*.
read: 50, 51*.
read_font_info: 58, 98*.
read_in: 17*.
read_tfm_word: 50, 60, 62*, 64, 68*.
read_two_halves: 60*.
read_two_halves_end: 60*.
real: 114, 134, 138*, 168, 229*.
rem_byte: 55*, 56, 122, 210*.
remainder: 40, 41, 42.
rep: 43*.
reset: 47*.
reswitch: 6, 210, 218.
return: 6, 7.
rewrite: 47.
rewritebin: 47.
right_coords: 186, 190, 196, 197, 198, 199.
right_quotes: 78, 172.
right1: 21.
right2: 21.
right3: 21.
right4: 21, 22, 119, 121, 171, 209.
round: 114, 167, 170, 178, 179, 180, 209, 218, 222.
rule_code: 77, 154.
rule_ptr: 158, 159, 161, 173.
rule_size: 158, 159, 173, 176, 177, 178.
rule_slant: 134, 137, 173, 178.
rule_thickness: 154, 155, 156, 159.
s: 58, 116, 220.
save_c: 116.
sc: 53, 55, 56, 57, 64, 66, 68.
second_octant: 192, 196, 197, 198, 199.
send_it: 116, 119, 121.
set_char_0: 21.
set_char_1: 21.
set_cur_x: 116, 122, 123.
set_rule: 19, 21.
set1: 21, 22, 113.
set2: 21.
set4: 21.
seventh_octant: 192, 196, 197, 198, 199.
signed_quad: 51, 81, 84, 85, 165.
sixth_octant: 192, 196, 197, 198, 199.
sixty_four_cases: 215.
skip_byte: 42, 55, 120.
skip_nop: 85, 215.
skip0: 29, 30, 215.
skip1: 29, 30, 215.
skip2: 29, 30, 215.
skip3: 29, 30, 215.
slant: 44, 57, 68, 137, 169.
slant_fonts: 35, 39.
slant_complaint: 138, 178.
slant_font: 52, 58, 77, 97, 98, 100, 137, 154, 173.
slant_n: 134, 137, 178.
slant_reported: 134, 137, 138.
slant_unit: 134, 137, 178, 179, 180.
small_logo: 78, 172.
Sorry, I can't...: 138.
sp: 23.
space: 44, 57, 119, 184.
space_shrink: 44.
space_stretch: 44.
Special font subst...: 99.
stack_ptr: 116, 122, 123.
starting_col: 208, 214, 217, 218.
stderr: 8, 222.
stdin: 16.
stdout: 3, 16.
stop_flag: 42, 66, 120.
store_four_quarters: 62, 63, 66, 67.
store_scaled: 64, 66, 68.
str_number: 70, 71, 74, 80, 86, 92, 93, 96, 116, 140, 220.
str_pool: 70, 71, 73, 74, 75, 83, 92, 100, 111, 112, 116, 221.
str_ptr: 71, 74, 75, 77, 91, 101, 154, 219, 221.
str_room: 73, 83, 90, 101.
str_start: 70, 71, 72, 74, 75, 77, 83, 91, 92, 100, 112, 116, 219, 221.
strncpy: 222.
stringcast: 47.
suppress Regulation: 116, 117, 120, 122.
sw: 58, 64, 68.
system dependencies: 2, 3, 8, 11, 14, 16, 17, 26, 33, 45, 47, 50, 51, 52, 78, 86, 87, 88, 89, 90, 91, 92, 105, 107.
t: 220.
tag: 40, 41.
Tardy font change...: 154.
temp_x: 173, 174, 177, 178.
temp_y: 173, 174, 176, 178.
term_in: 16, 17.
terminal line_length: 5, 16, 17, 18, 94.
TeXfonts: 88.
text_char: 11, 12, 48.
text_file: 11.
tfm_ext: 78, 98.
tfm_file: 46, 47, 50, 58.
third_octant: 192, 196, 197, 198, 199.
thirty_two_cases: 215.
three_x_height: 183, 184, 202.
timestamp: 172, 220, 221.
title_code: 77, 154.
title_font: 52, 58, 77, 78, 97, 98, 100, 115, 154, 172.
Too many labels: 74, 141.
Too many strings: 73, 91.
top: 43.
top_coords: 153, 150, 151, 152, 172, 190, 194, 202, 215, 221.
total_pages: 102, 103, 115, 164, 172, 199.
true: 7, 90, 95, 96, 97, 122, 146, 147, 148, 151, 152, 172, 190, 194, 202, 215, 221.
twin: 148, 149, 150, 151, 152, 192.
two_to_the: 126, 127, 128, 129, 206.
terminate: 94.
terminate: 8, 115.
unity: 9, 62, 114, 137, 168, 169, 170.
unsc_slant_ratio: 168, 169, 170, 209.
unsc_x_ratio: 168, 169, 170, 209, 218.
unsc_y_ratio: 168, 169, 170, 218.
update_terminal: 16, 17, 164.
usage: 222.
usage_help: 222.
use_log: 172, 220, 221.
v: 84, 98, 218.
val: 223, 224, 225, 228, 231.
Vanishing pixel size: 169.
version_string: 3.
uppp: 31.
WEB: 72.
web2c: 52.
white: 25.
widest_row: 5, 165, 211, 218.
width_base: 53, 55, 61, 63, 64, 69.
width_index: 40, 53.
write: 3, 107.
Wrong ID: 221.
w0: 21.
w1: 21.
w2: 21.
w3: 21.
w4: 21.
x_height: 44, 57, 184.
x_left: 145, 146, 147.
x_offset: 154, 155, 156, 167.
x_offset_code: 77, 154.
x_right: 145, 146, 147.
xchr: 12, 13, 14, 15, 92.
xclause: 7.
(Add a full set of $k$-bit characters 128) Used in section 126.
(Add more rows to $a$, until 12-bit entries are obtained 213) Used in section 218.
(Add special $k$-bit characters of the form $X_0 \ldots X_0$ 129) Used in section 126.
(Adjust the maximum page width 203) Used in section 164*.
(Advance to the next row that needs to be typeset; or return, if we’re all done 217) Used in section 218.
(Carry out a ligature operation, updating the cursor structure and possibly advancing $k$; goto continue if the cursor doesn’t advance, otherwise goto done 122) Used in section 120.
(Compute the octant code for floating label $p$ 192) Used in section 191.
(Constants in the outer block 5) Used in section 3*.
(Declare the procedure called load_fonts 98) Used in section 111*.
(Define the option table 223*, 224*, 225*, 228*, 231*) Used in section 222*.
(Define parse_arguments 222*) Used in section 3*.
(Empty the last bytes out of dwi_buf 109*) Used in section 115*.
(Enter a dot for label $p$ in the rectangle list, and typeset the dot 188) Used in section 187.
(Enter a prescribed label for node $p$ into the rectangle list, and typeset it 190) Used in section 189.
(Find nearest dots, to help in label positioning 191) Used in section 181.
(Find non-overlapping coordinates, if possible, and goto found; otherwise set place_label ← false and return 195) Used in section 194.
(Finish reading the parameters of the boc 165) Used in section 164*.
(Finish the DVI file and goto final_end 115*) Used in section 219*.
(Get online special input 99) Used in section 98.
(Get ready to convert METAFONT coordinates to DVI coordinates 170*) Used in section 164*.
(Globals in the outer block 12, 16*, 18, 37, 46, 48*, 49, 53, 71, 76, 80, 86, 87, 93, 96, 102, 105, 117, 127, 134, 140, 149, 155, 158, 160, 166, 168, 174, 182, 183, 207, 211, 212, 220, 226*, 229*) Used in section 3*.
(If the keyword in buffer[1 . . l] is known, change c and goto done 83) Used in section 82.
(If there’s a ligature or kern at the cursor position, update the cursor data structures, possibly advancing $k$; continue until the cursor wants to move right 120) Used in section 116.
(Initialize global variables that depend on the font data 137, 169, 175, 184, 205, 206) Used in section 98.
(Initialize the option variables 227*, 230*) Used in section 222*.
(Initialize the strings 77, 78*, 88*) Used in section 219*.
(Initialize variables for the next character 144, 156, 161) Used in section 219*.
(Look for overlaps in node $q$ and its predecessors 147) Used in section 145.
(Look for overlaps in the successors of node $q$ 146) Used in section 145.
(Make final adjustments and goto done 69) Used in section 59.
(Move the cursor to the right and goto continue, if there’s more work to do in the current word 123) Used in section 116.
(Move to column $j$ in the DVI output 209) Used in section 208.
(Output a horizontal rule 177) Used in section 173.
(Output a vertical rule 176) Used in section 173.
(Output all attachable labels 193) Used in section 181.
(Output all dots 187) Used in section 181.
(Output all labels for the current character 181) Used in section 164*.
(Output all overflow labels 200) Used in section 181.
(Output all prescribed labels 189) Used in section 181.
(Output all rules for the current character 173) Used in section 164*.
(Output the equivalent of $k$ copies of character $v$ 210) Used in section 208.
(Output the font name whose internal number is $f$ 112) Used in section 111*.
(Output the bop and the title line 172) Used in section 164*.
(Override the offsets 157) Used in section 154.
(Paint $k$ bits and read another command 216) Used in section 215*.
(Process a character 164*) Used in section 219*.
(Process a no-op command 154) Used in section 219*.

(Process the preamble 221) Used in section 219*.
(Put the bits for the next row, times l, into a 214) Used in section 213.
(Read and check the font data; abend if the TFM file is malformed; otherwise goto done 59) Used in section 58.
(Read and process GF commands until coming to the end of this row 215*) Used in section 214.
(Read box dimensions 64) Used in section 59.
(Read character data 63) Used in section 59.
(Read extensible character recipes 67) Used in section 59.
(Read font parameters 68) Used in section 59.
(Read ligature/kern program 66) Used in section 59.
(Read the next k characters of the GF file; change c and goto done if a keyword is recognized 82) Used in section 81*.
(Read the TFM header 62*) Used in section 59.
(Read the TFM size fields 60) Used in section 59.
(Remove all rectangles from list, except for dots that have labels 201) Used in section 200.
(Replace z by z' and compute α, β 65) Used in section 64.
(Scan the file name in the buffer 95) Used in section 94*.
(Search buffer for valid keyword; if successful, goto found 100) Used in section 99.
(Search for the nearest dot in nodes following p 151) Used in section 150.
(Search for the nearest dot in nodes preceding p 152) Used in section 150.
(Set initial values 13, 14*, 15, 54, 97, 103, 106, 118*, 126, 142) Used in section 3*.
(Store a label 163) Used in section 154.
(Store a rule 159) Used in section 154.
(Store a title 162) Used in section 154.
(Try the first choice for label direction 196) Used in section 195.
(Try the fourth choice for label direction 199) Used in section 195.
(Try the second choice for label direction 197) Used in section 195.
(Try the third choice for label direction 198) Used in section 195.
(Try to output a diagonal rule 178) Used in section 173.
(Typeset a space in font f and advance k 119) Used in section 116.
(Typeset an overflow label for p 202) Used in section 200.
(Typeset character cur,l, if it exists in the font; also append an optional kern 121) Used in section 116.
(Typeset the pixels of the current row 208) Used in section 218.
(Update the pixels of the current row 208) Used in section 218.
(Use size fields to allocate font information 61) Used in section 59.
(Vertically typeset p copies of character k + 1 180) Used in section 178.
(Vertically typeset q copies of character k 179) Used in section 178.