The GFtype processor

(Version 3.1, March 1991)
1* Introduction. The GFtype utility program reads binary generic-font ("GF") files that are produced by font compilers such as METAFONT, and converts them into symbolic form. This program has three chief purposes: (1) It can be used to look at the pixels of a font, with one pixel per character in a text file; (2) it can be used to determine whether a GF file is valid or invalid, when diagnosing compiler errors; and (3) it serves as an example of a program that reads GF files correctly, for system programmers who are developing GF-related software.

The original version of this program was written by David R. Fuchs in March, 1984. Donald E. Knuth made a few modifications later that year as METAFONT was taking shape.

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

\[
\begin{align*}
\text{define } \text{my\_name} & \equiv \text{`gftype'} \\
\text{define } \text{banner} & \equiv \text{`This\_is\_GFtype,\_Version\_3.1'} \quad \text{printed when the program starts}
\end{align*}
\]

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

\[
\begin{align*}
\text{define } \text{print}(\#) & \equiv \text{write}(\text{stdout}, \#) \\
\text{define } \text{print\_ln}(\#) & \equiv \text{write\_ln}(\text{stdout}, \#) \\
\text{define } \text{print\_nl} & \equiv \text{write\_ln}(\text{stdout})
\end{align*}
\]

program GF\_type(gf\_file, output);
\textbf{const} \{ Constants in the outer block 5* \}
\textbf{type} \{ Types in the outer block 8 \}
\textbf{var} \{ Globals in the outer block 4* \}
\begin{itemize}
\item Define \textit{parse\_arguments} 73*
\end{itemize}

\textbf{procedure} initialize; \{ this procedure gets things started properly \}
\textbf{var} \begin{itemize}
\item \textit{i} : \textit{integer}; \{ loop index for initializations \}
\item \textit{bound\_default} : \textit{integer}; \{ temporary for setup \}
\item \textit{bound\_name} : \textit{const\_cstring}; \{ temporary for setup \}
\end{itemize}
\begin{itemize}
\item \textit{kpse\_set\_program\_name(\textit{argv}[0], \textit{my\_name}); kpse\_init\_pro}g(\`GFTYPE\', 0, nil, nil);
\item \textit{parse\_arguments}; \textit{print}(\textit{banner}); \textit{print\_ln}(\textit{version\_string}); \{ Set initial values 6* \}
\end{itemize}
\end{itemize}

4* This module is deleted, because it is only useful for a non-local goto, which we can't use in C. Instead, we define parameters settable at runtime.

\begin{itemize}
\item \textbf{line\_length} : \textit{integer}; \{ \textit{xxx} strings will not produce lines longer than this \}
\item \textit{max\_rows} : \textit{integer}; \{ largest possible vertical extent of pixel image array \}
\item \textit{max\_cols} : \textit{integer}; \{ largest possible horizontal extent of pixel image array \}
\item \textit{max\_row} : \textit{integer}; \{ current vertical extent of pixel image array \}
\item \textit{max\_col} : \textit{integer}; \{ current horizontal extent of pixel image array \}
\end{itemize}

See also sections 10, 21, 23, 25*, 35, 37*, 41, 46, 54, 62, and 67.

This code is used in section 3*. 
Three parameters can be changed at run time to extend or reduce \texttt{GFtype}'s capacity. Note that the total number of bits in the main \texttt{image\_array} will be

\[(\text{max\_row} + 1) \times (\text{max\_col} + 1).\]

(META\textsc{font}'s full pixel range is rarely implemented, because it would require 8 megabytes of memory.)

\begin{verbatim}
define def\_line\_length = 500  { default \texttt{line\_length} value }
define max\_image = 8191  { largest possible extent of META\textsc{font}'s pixel image array }
\end{verbatim}

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

\begin{verbatim}
inf\_line\_length = 20; sup\_line\_length = 1023;
This code is used in section 3*.
\end{verbatim}

Here are some macros for common programming idioms.

\begin{verbatim}
define incr(#) \equiv # \leftarrow # + 1  { increase a variable by unity }
define decr(#) \equiv # \leftarrow # - 1  { decrease a variable by unity }
define negate(#) \equiv # \leftarrow -#  { change the sign of a variable }
define const\_chk(#) \equiv 
   begin if # < inf @&# then # \leftarrow inf @&#
   else if # > sup @&# then # \leftarrow sup @&#
   end  { setup\_bound\_var stuff duplicated in tex.ch. }
define setup\_bound\_var(#) \equiv bound\_default \leftarrow #; setup\_bound\_var\_end
define setup\_bound\_var\_end(#) \equiv bound\_name \leftarrow #; setup\_bound\_var\_end\_end
define setup\_bound\_var\_end\_end(#) \equiv setup\_bound\_variable(address\_of(#), bound\_name, bound\_default);
\end{verbatim}

\texttt{(Set initial values 6*)} \equiv

\begin{verbatim}
\{ See comments in tex.ch for why the name has to be duplicated. \}
setup\_bound\_var(def\_line\_length)(`line\_length')(line\_length);
   \{ xxx strings will not produce lines longer than this \}
setup\_bound\_var(max\_image)(`max\_rows')(max\_rows);
   \{ largest allowed vertical extent of pixel image array \}
setup\_bound\_var(max\_image)(`max\_cols')(max\_cols);
   \{ largest allowed horizontal extent of pixel image array \}
const\_chk(line\_length);
if max\_rows > max\_image then max\_rows \leftarrow max\_image;
if max\_cols > max\_image then max\_cols \leftarrow max\_image;
image\_array \leftarrow nil;
\end{verbatim}

See also sections 11, 12, 26*, 47, and 63.

This code is used in section 3*.

If the \texttt{GF} file is badly malformed, the whole process must be aborted; \texttt{GFtype} 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 we might want to \texttt{abort} the program with an error message.

\begin{verbatim}
define abort(#) \equiv
   begin write\_ln(stderr, #); uexit(1);
   end
define bad\_gf(#) \equiv abort(`Bad\_GF\_file:\u201c, #, `\')
\end{verbatim}
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 \texttt{GFtype}. So we shall assume that the Pascal system being used for \texttt{GFtype} has a character set containing at least the standard visible characters of ASCII code ("!" through "~").

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 char ≡ 0..255    { the data type of characters in text files }
  define text_char ≡ char  { ordinal number of the smallest element of text_char }
  define last_text_char = 127   { ordinal number of the largest element of text_char }

  text_file = packed file of text_char;
\end{verbatim}
In C, we do path searching based on the user’s environment or the default path.

procedure open_gf_file;  { prepares to read packed bytes in gf_file }
    begin
        gf_file ← kpse_open_file(cmdline(optind), kpse_gf_format); cur_loc ← 0;
        ⟨ Print all the selected options 34* ⟩;
    end;
Optional modes of output. \texttt{GFtype} will print different quantities of information based on some options that the user must specify: We set \texttt{wants\_mnemonics} if the user wants to see a mnemonic dump of the \texttt{GF} file; and we set \texttt{wants\_pixels} if the user wants to see a pixel image of each character.

When \texttt{GFtype} begins, it engages the user in a brief dialog so that the options will be specified. This part of \texttt{GFtype} requires nonstandard Pascal constructions to handle the online interaction; so it may be preferable in some cases to omit the dialog and simply to produce the maximum possible output (\texttt{wants\_mnemonics} = \texttt{wants\_pixels} = \texttt{true}). On other hand, the necessary system-dependent routines are not complicated, so they can be introduced without terrible trauma.

\begin{verbatim}
⟨ Globals in the outer block 4* ⟩ +≡

\texttt{wants\_mnemonics: c\_int\_type; \{ controls mnemonic output \}}
\texttt{wants\_pixels: c\_int\_type; \{ controls pixel output \}}

⟨ Set initial values 6* ⟩ +≡

26* There is no terminal input. The options for running this program are offered through command line options.

29* During the dialog, extensions of \texttt{GFtype} might treat the first blank space in a line as the end of that line. Therefore \texttt{input\_ln} makes sure that there is always at least one blank space in \texttt{buffer}.

(\text{This routine is more complex than the present implementation needs, but it has been copied from \texttt{DVItype} so that system-dependent changes that worked before will work again.})

30* This was so humdrum that we got rid of it. (module 30)

31* The dialog procedure module is eliminated. (module 31)

32* So is its first part. (module 32)

33* So is its second part. (module 33)

34* After the command-line switches have been processed, we print the options so that the user can see what \texttt{GFtype} thought was specified.

\begin{verbatim}
⟨ Print all the selected options 34* ⟩ ≡

\texttt{print("Options selected: Mnemonic output =", \texttt{wants\_mnemonics});}
\texttt{if \texttt{wants\_mnemonics} then print("true") else print("false");}
\texttt{print("; Pixel output =", \texttt{wants\_pixels});}
\texttt{if \texttt{wants\_pixels} then print("true") else print("false");}
\texttt{print\_ln(".");}
\end{verbatim}

This code is used in section 22*. 

In order to allow different systems to change the \textit{image} array easily from row-major order to column-major order (or vice versa), or to transpose it top and bottom or left and right, we declare and access it as follows.

\begin{verbatim}
define image \equiv image_array[m + (max_col + 1) \times n]
\end{verbatim}

\begin{verbatim}
\langle\text{Globals in the outer block}\rangle \equiv image_array: \uparrow \text{pixel};
\end{verbatim}

A \texttt{boc} command has parameters \texttt{min\_m}, \texttt{max\_m}, \texttt{min\_n}, and \texttt{max\_n} that define a rectangular subarray in which the pixels of the current character must lie. The program here computes limits on \texttt{GF\_type}'s modified \texttt{m} and \texttt{n} variables, and clears the resulting subarray to all \texttt{white}.

(There may be a faster way to clear a subarray on particular systems, using nonstandard extensions of \texttt{Pascal}.)

\begin{verbatim}
\langle\text{Clear the image}\rangle \equiv
\begin{align*}
\text{begin} & \quad \text{max\_col} \leftarrow \text{max\_m\_stated} - \text{min\_m\_stated} - 1; \\
& \quad \text{if } \text{max\_col} > \text{max\_cols} \text{ then } \text{max\_col} \leftarrow \text{max\_cols}; \\
& \quad \text{max\_row} \leftarrow \text{max\_n\_stated} - \text{min\_n\_stated}; \\
& \quad \text{if } \text{max\_row} > \text{max\_rows} \text{ then } \text{max\_row} \leftarrow \text{max\_rows}; \\
& \quad \text{if } (\text{max\_row} \geq 0) \land (\text{max\_col} \geq 0) \text{ then } \text{image\_array} \leftarrow \text{xalloc\_array} \langle \text{pixel}, \text{max\_col}, \text{max\_row} \rangle;
\end{align*}
\end{verbatim}

This code is used in section 71.

With \texttt{image\_array} allocated dynamically these are the same.

\begin{verbatim}
define max\_subrow \equiv \text{max\_row} \quad \{ \text{vertical size of current subarray of interest} \}
define max\_subcol \equiv \text{max\_col} \quad \{ \text{horizontal size of current subarray of interest} \}
\end{verbatim}

As we paint the pixels of a character, we will record its actual boundaries in variables \texttt{max\_m\_observed} and \texttt{max\_n\_observed}. Then the following routine will be called on to output the image, using blanks for \texttt{white} and asterisks for \texttt{black}. Blanks are emitted only when they are followed by nonblanks, in order to conserve space in the output. Further compaction could be achieved on many systems by using tab marks.

An integer variable \texttt{b} will be declared for use in counting blanks.

\begin{verbatim}
\langle\text{Print the image}\rangle \equiv
\begin{align*}
\text{begin} & \quad \text{(Compare the subarray boundaries with the observed boundaries)} \quad \text{42);} \\
& \quad \text{if } \text{max\_subcol} \geq 0 \text{ then } \quad \{ \text{there was at least one \texttt{paint} command} \} \\
& \quad \quad \langle\text{Print asterisk patterns for rows 0 to \texttt{max\_subrow}}\rangle \quad \text{43} \rangle \\
& \quad \quad \text{else } \text{print\_ln}(\langle \text{The character is entirely blank.} \rangle); \\
& \quad \quad \text{if } (\text{max\_row} \geq 0) \land (\text{max\_col} \geq 0) \text{ then} \\
& \quad \quad \quad \text{begin } \text{libc\_free}(\text{image\_array}); \text{ image\_array} \leftarrow \text{nil};
\end{align*}
\end{verbatim}

This code is used in section 69.
We steal the following routine from METAFONT.

```
define unity ≡ '200000 { 2^16, represents 1.00000 }
```

```
procedure print_scaled(s : integer); { prints a scaled number, rounded to five digits }

var delta: integer; { amount of allowable inaccuracy }

begin if s < 0 then
    begin print(‘−’); negate(s); { print the sign, if negative }
    end;
    print(s div unity : 1); { print the integer part }
    s ← 10 * (s mod unity) + 5;

    if s ≠ 5 then
        begin delta ← 10; print(‘.’);
            repeat if delta > unity then s ← s − '100000 − (delta div 2); { round the final digit }
                print(xchr[ord(‘0’) + (s div unity)]);
            until s ≤ delta;
        end;
    end;
end;
```

Before we get into the details of do_char, it is convenient to consider a simpler routine that computes the first parameter of each opcode.

```
define four_cases(♯) ≡ ♯, ♭ + 1, ♭ + 2, ♭ + 3
define eight_cases(♯) ≡ four_cases(♯), four_cases(♯ + 4)
define sixteen_cases(♯) ≡ eight_cases(♯), eight_cases(♯ + 8)
define thirty_two_cases(♯) ≡ sixteen_cases(♯), sixteen_cases(♯ + 16)
define thirty_two_cases(♯) ≡ thirty_two_cases(♯), four_cases(♯ + 32), ♭ + 36
define sixty_four_cases(♯) ≡ thirty_two_cases(♯), thirty_two_cases(♯ + 32)
```

```
function first_par(o : eight_bits): integer;

begin case o of
    sixty_four_cases(paint_0): first_par ← o − paint_0;
paint1, skip1, char_loc, char_loc + 1, xxx1: first_par ← get_byte;
paint1 + 1, skip1 + 1, xxx1 + 1: first_par ← get_two_bytes;
paint1 + 2, skip1 + 2, xxx1 + 2: first_par ← get_three_bytes;
xxx1 + 3, yyy: first_par ← signed_quad;
boc, boc1, coe, skip0, no_op, pre, post, post_post, undefined_commands: first_par ← 0;
sixty_four_cases(new_row_0): first_par ← o − new_row_0;
sixty_four_cases(new_row_0 + 64): first_par ← o − new_row_0;
three_seven_cases(new_row_0 + 128): first_par ← o − new_row_0;
othercases abort(‘internal_error’)
endcases;
end;
```
The multiway switch in first_par, above, was organized by the length of each command; the one in do_char is organized by the semantics.

\[\langle\text{Start translation of command } o \text{ and } \text{goto the appropriate label to finish the job } 51^*\rangle \equiv\]
\[\text{if } o \leq paint1 + 3 \text{ then } \langle\text{Translate a sequence of paint commands, until reaching a non-paint } 56\rangle;\]
\[\text{case } o \text{ of}\]
\[\text{four_cases} (\text{skip0}): \langle\text{Translate a skip command } 60\rangle;\]
\[\text{sixty_four_cases} (\text{new_row_0}): \langle\text{Translate a new_row command } 59\rangle;\]
\[\text{sixty_four_cases} (\text{new_row_0} + 64): \langle\text{Translate a new_row command } 59\rangle;\]
\[\text{thirty_seven_cases} (\text{new_row_0} + 128): \langle\text{Translate a new_row command } 59\rangle;\]
\[\langle\text{Cases for commands no_op, pre, post, post_post, boc, and eoc } 52\rangle\]
\[\text{four_cases} (\text{xxx1}): \langle\text{Translate an xxx command } 53\rangle;\]
\[\text{yyy}: \langle\text{Translate a yyy command } 55\rangle;\]
\[\text{other_cases} \text{ error} (\text{`undefined} o \text{command}', o : 1, '!' )\]
\[\text{endcases}\]

This code is used in section 50.
The main program. Now we are ready to put it all together. This is where GFtype starts, and where it ends.

begin initialize; { get all variables initialized }
 ⟨Process the preamble 68⟩;
 ⟨Translate all the characters 69⟩;
 print nl; read_postamble; print(“The file had, total_chars : 1, character”);
 if total_chars ≠ 1 then print(“s”);
 print ln(“altogether.”);
end.
§73  **System-dependent changes.** Parse a Unix-style command line.

```pascal
define argument_is(#) ≡ (strcmp(long_options[option_index].name, #) = 0)
define do_nothing ≡ { empty statement }
(Define parse_arguments 73*) ≡

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;
  optarg_return_val: integer; option_index: c_int_type; current_option: 0 .. n_options;
  begin (Define the option table 74*)
  repeat optarg_return_val ← getopt_long_only(argc, argv, ` `, long_options, address_of(option_index));
    if optarg_return_val = -1 then
      begin do_nothing; { End of arguments; we exit the loop below. }
      end
    else if optarg_return_val = `?` then
      begin usage(my_name);
      end
    else if argument_is(`help`) then
      begin usage_help(GFTYPE_HELP, nil);
      end
    else if argument_is(`version`) then
      begin print_version_and_exit(banner, nil, `D.R. Fuchs`, nil);
      end; { Else it was a flag. }
  until optarg_return_val = -1; { Now optind is the index of first non-option on the command line. We
    must have one remaining argument. }
  if (optind + 1 ≠ argc) then
    begin writeLn(stderr, my_name, ` Need exactly one file argument. `); usage(my_name);
    end;
  end;
```

This code is used in section 3*.

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

```pascal
(Define the option table 74*) ≡
  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 75*, 76*, 77*, and 78*.

This code is used in section 73*.

§75*  Another of the standard options.

```pascal
(Define the option table 74*) +≡
  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);
```

§76*  Translate commands?

```pascal
(Define the option table 74*) +≡
  long_options[current_option].name ← `mnemonics`; long_options[current_option].has_arg ← 0;
  long_options[current_option].flag ← address_of(wants_mnemonics);
  long_options[current_option].val ← 1; incr(current_option);
```
77* Show pixels?
\langle Define the option table \rangle +≡
\quad long_options[\text{current\_option}].name ← 'images'; long_options[\text{current\_option}].has_arg ← 0;
\quad long_options[\text{current\_option}].flag ← address\_of\(\text{wants\_pixels}\); long_options[\text{current\_option}].val ← 1;
\quad \text{incr}(\text{current\_option});

78* An element with all zeros always ends the list.
\langle Define the option table \rangle +≡
\quad long_options[\text{current\_option}].name ← 0; long_options[\text{current\_option}].has_arg ← 0;
\quad long_options[\text{current\_option}].flag ← 0; long_options[\text{current\_option}].val ← 0;
79* 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, 6, 7, 9, 22, 25, 26, 27, 29, 30, 31, 32, 33, 34, 37, 38, 39, 40, 45, 48, 51, 66, 73, 74, 75, 76, 77, 78, 79.

- help: 74*
- version: 75*
a: 24, 67.
abort: 7* 48*.
address_of: 6* 73* 76* 77*.
aok: 49.
argc: 73*.
argument_is: 73*.
argv: 3* 73*.
ASCII_code: 8, 10.
b: 24, 67.
backpointer...should be p: 61.
backpointers: 18.
Bad GF file: 7*.
bad_char: 53, 54.
bad_gf: 7* 50, 64, 68, 69.
banner: 1* 3* 73*.
black: 14, 15, 35, 36, 40* 57, 58, 59.
boc: 13, 15, 16, 17, 18, 38* 42, 44, 48* 49, 52, 69, 71.
boc occurred before eoc: 52.
boc1: 15, 16, 48* 52, 69.
boolean: 36, 49, 54.
bound_default: 3* 6*.
bound_name: 3* 6*.
break: 28.
buffer: 29*.
byte n is not boc: 69.
byte_file: 20, 21.
c: 24, 61, 67.
c_int_type: 25* 73*.
char: 9*.
char ended unexpectedly: 69.
char_loc: 15, 16, 18, 48* 65.
char_loc0: 15.
char_ptr: 46, 47, 64, 65, 71.
character location should be...: 65.
character_code: 46, 71.
check sum: 17.
check_sum: 61, 62.
Chinese characters: 18.
chr: 9* 10, 12.
cmdline: 22*.
const_chk: 6*.
const_cstring: 3*.
cs: 17.
cur_loc: 22* 23, 24, 50, 61, 64, 65, 69, 71.
current_option: 73* 74* 75* 76* 77* 78*.
d: 24.
decr: 6* 43, 53, 68.
def_line_length: 5* 6*.
del_m: 15.
del_n: 15.
delta: 45*.
design size: 17.
design_size: 61, 62.
dm: 15.
do_char: 44, 48* 49, 51* 69.
do_nothing: 73*.
ds: 17.
duplicate locator...: 65.
dz: 15, 18.
dy: 15, 18.
eight_bits: 20, 24, 48* 49.
eight_cases: 48*.
else: 2.
end: 2.
endcases: 2.
ecoc: 13, 15, 16, 17, 48* 52.
eof: 24, 50, 64.
error: 50, 51* 52, 61, 64, 65, 71.
false: 36, 49, 53.
First byte isn’...: 68.
first_par: 48* 50, 51* 65.
first_text_char: 9* 12.
flag: 74* 75* 76* 77* 78*.
four_cases: 48* 51*.
Fuchs, David Raymond: 1* 19.
get_byte: 24, 48* 50, 53, 64, 65, 68, 71.
get_three_bytes: 24, 48*.
get_two_bytes: 24, 48*.
getopt_long_only: 73*.
getopt_return_val: 73*.
getopt_struct: 73*.
gf_file: 3* 21, 22* 23, 24, 50, 64.
gf_id_byte: 15, 64, 68.
gf_prev_ptr: 46, 61, 69, 71.
GF_type: 3*.
GFTYPE_HELP: 73*.
has_arg: 74* 75* 76* 77* 78*.
hppp: 17, 61, 62.
i: 3*.
identification byte should be n: 64, 68.
image: 37* 43, 58.
image_array: 5* 6* 37* 38* 39* 40*.
inr: 6* 24, 43, 53, 58, 59, 71, 74* 75* 76* 77*.
inf: 6*.
inf_line_length: 5
initialize: 3, 66
input_ln: 29
integer: 3, 4, 23, 24, 35, 41, 45, 46, 48, 49, 61, 62, 67, 73
k: 61.
Knuth, Donald Ervin: 1
kpse_gf_format: 22
kpse_init_prog: 3
kpse_open_file: 22
kpse_set_program_name: 3
k: 67.
last_text_char: 9, 12
libc_free: 40
line_length: 4, 5, 6, 53
long_options: 73, 74, 75, 76, 77, 78
m: 35, 61
max_col: 4, 5, 37, 38, 39, 40, 42
max_cobs: 4, 6, 38
max_image: 5, 6
max_int: 63.
max_m: 15, 17, 38
max_m_observed: 40, 41, 42, 57, 71, 72
max_m_overall: 41, 61, 63, 72
max_m_stated: 38, 41, 61, 71, 72
max_n: 15, 17, 35, 38
max_n_observed: 40, 41, 42, 69, 72
max_n_overall: 41, 61, 63, 72
max_n_stated: 38, 41, 43, 59, 60, 61, 71, 72
max_row: 4, 5, 38, 39, 40, 42
max_rows: 4, 6, 38
max_subcol: 39, 40, 42, 43, 58
max_subset: 39, 40, 42, 43, 58
min_m: 15, 17, 35, 38
min_m_overall: 41, 61, 63, 72
min_m_stated: 38, 41, 43, 61, 71, 72
min_n: 15, 17, 38
min_n_overall: 41, 61, 63, 72
min_n_stated: 38, 41, 61, 71, 72
missing locator...: 64.
my_name: 1, 3, 73
n: 35
n_options: 73
name: 73, 74, 75, 76, 77, 78
negate: 6, 45
new_row_0: 15, 16, 48, 51
new_row_1: 15
new_row_164: 15
nl_error: 50, 53
no_op: 15, 16, 18, 48, 52, 65, 70.
non-ASCII character...: 53.
not enough signature bytes...: 64.
o: 49, 67.
open_gf_file: 22, 68
option: 22, 73
option_index: 73
Options selected: 34
ord: 10, 45
oriental characters: 18
otherwise: 2
others: 2
output: 3
p: 49, 61, 67
paint: 56
paint_switch: 14, 29, 57, 58, 59, 60, 71
paint_0: 15, 16, 48
paint1: 15, 16, 48, 51, 56
paint2: 15
paint3: 15
parse_arguments: 3, 73
pix_ratio: 61, 62, 65
pixel: 35, 36, 37, 38
post: 13, 15, 16, 17, 19, 48, 52, 61, 62, 69.
post_loc: 61, 62, 64.
post_post: 15, 16, 17, 19, 48, 52, 64.
postamble command within...: 52
postamble pointer should be...: 64.
Postamble starts at byte n: 61.
pre: 13, 15, 16, 48, 52, 68.
preamble command within...: 52
previous character...: 71, 72
print: 3, 34, 43, 45, 50, 53, 55, 56, 57, 59, 60, 61, 65, 66, 68, 69, 71
print_0: 23, 43, 50, 52, 53, 66, 69
print_scaled: 43, 55, 61, 65
print_version_and_exit: 73
proofing: 18
q: 49, 61, 67
read: 24.
read_postamble: 61, 66
real: 62
round: 65.
s: 45
scaled: 15, 17, 18
setup_bound_var: 6
setup_bound_var_end: 6
setup_bound_var_end: 6
setup_bound_variable: 6
should be postpost: 64.
show_label: 50
show_mnemonic: 50, 52, 53, 55, 59, 60, 70.
signature...should be...: 64.  
signed_quad: 24, 48, 61, 64, 65, 71.  
sixteen_cases: 48*.  
sixty_four_cases: 48*, 51*.  
skip0: 15, 16, 48*, 51*.  
skip1: 15, 16, 48*, 60.  
skip2: 15.  
skip3: 15.  
start_op: 50, 56, 70.  
stderr: 7*, 73*.  
stdout: 3*.  
strcmp: 73*.  
string of negative length: 53.  
sup: 6*  
sup_line_length: 5*.  
system dependencies: 2, 7*, 9*, 19, 20, 24, 25*, 28, 29*, 36, 37, 38*, 40*.  
term_out: 28.  
text_char: 9*, 10.  
text_file: 9*.  
The character is too large...: 42.  
The file ended prematurely: 50.  
The file had n characters...: 66*.  
thirty_seven_cases: 48*, 51*.  
thirty_two_cases: 48*.  
This pixel’s lower...: 43.  
This pixel’s upper: 43.  
total_chars: 46, 47, 66, 71.  
true: 25*, 36, 49, 53.  
v: 61.  
uexit: 7*.  
undefined command: 51*.  
undefined_commands: 16, 48*.  
unity: 45*, 65.  
update_terminal: 28.  
usage: 73*.  
usage_help: 73*.  
v: 61.  
val: 74*, 75*, 76*, 77*, 78*.  
version_string: 3*.  
uppp: 17, 61, 62.  
w: 61.  
wants_mnemonics: 25*, 34*, 50, 53, 55, 56, 57, 59, 60, 71, 76*.  
wants_pixels: 25*, 34*, 57, 69, 71, 77*.  
white: 15, 35, 36, 38*, 40*, 43, 57, 58, 60, 71.  
write: 3*.  
write_ln: 3*, 7*, 73*.  
xalloc_array: 38*.  
xchr: 10, 11, 12, 45*, 53, 68.  
xord: 10, 12.  
xxx1: 15, 16, 48*, 51*, 70.
Cases for commands *no_op, pre, post, post_post, boc,* and *eoc* 52) Used in section 51*.

Clear the image 38*) Used in section 71.

Compare the subarray boundaries with the observed boundaries 42) Used in section 40*.

Constants in the outer block 5*) Used in section 3*.

Define the option table 74*, 75*, 76*, 77*, 78*) Used in section 73*.

Define *parse_arguments 73* Used in section 73*.


Make sure that the end of the file is well-formed 64) Used in section 61.

Paint pixels m – p through m – 1 in row n of the subarray 58) Used in section 57.

Paint the next p pixels 57) Used in section 56.

Pass a *boc command 71) Used in section 69.

Pass an *eoc command 72) Used in section 69.

Pass *no_op, xxx and yyy commands 70) Used in section 69.

Print all the selected options 34*) Used in section 22*.

Print asterisk patterns for rows 0 to *max_subrow 43) Used in section 40*.

Print the image 40*) Used in section 69.

Process the character locations in the postamble 65) Used in section 61.

Process the preamble 68) Used in section 66*.

Set initial values 6*, 11, 12, 26*, 47, 63) Used in section 3*.

Start translation of command o and goto the appropriate label to finish the job 51*) Used in section 50.

Translate a sequence of *paint commands, until reaching a non-paint 56) Used in section 51*.

Translate a *new_row command 59) Used in sections 51*, 51*, and 51*.

Translate a skip command 60) Used in section 51*.

Translate a yyy command 55) Used in sections 51* and 70.

Translate all the characters 69) Used in section 66*.

Translate an *xxx command 53) Used in sections 51* and 70.

Translate the next command in the GF file; goto 9999 if it was eoc; goto 9998 if premature termination is needed 50) Used in section 49.

Types in the outer block 8, 9*, 20, 36) Used in section 3*. 