The GFtype processor

(Version 3.1, March 1991)

Introduction ................................................................. 1 102
The character set .......................................................... 8 104
Generic font file format .................................................. 13 105
Input from binary files ................................................... 20 105
Optional modes of output ................................................ 25 106
The image array ............................................................. 35 107
Translation to symbolic form ........................................... 44 108
Reading the postamble .................................................... 61 110
The main program .......................................................... 66 110
System-dependent changes .............................................. 73 111
Index ............................................................................ 79 113

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. 'METAOFONT' is a trademark of Addison-Wesley Publishing Company.

March 17, 2021 at 13:05
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.

```plaintext
define my_name ≡ "gftype" define banner ≡ "This is GFtype, Version 3.1"  { printed when the program starts }
```

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

```plaintext
define print(#) ≡ write(stdout,#) define print_lin(#) ≡ write_in(stdout,#) define print_nl ≡ write_in(stdout)
```

**program GF_type(gf_file, output);**

```plaintext
const ⟨Constants in the outer block 5*⟩
type ⟨Types in the outer block 8⟩
var ⟨Globals in the outer block 4*⟩
  ⟨Define parse_arguments 73*⟩
procedure initialize;  { this procedure gets things started properly }
  var i: integer;  { loop index for initializations }
    bound_default: integer;  { temporary for setup }
    bound_name: const cstring;   { temporary for setup }
  begin kpse_set_program_name(argv[0], my_name); kpse_init_prog(\"GFTYPE\", 0, nil, nil);
    parse_arguments; print(banner); print_lin(version_string);  ⟨Set initial values 6*⟩
  end;
```

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.

```plaintext
(Globals in the outer block 4*) ≡
  line_length: integer;  { xxx strings will not produce lines longer than this }
  max_rows: integer;  { largest possible vertical extent of pixel image array }
  max_cols: integer;  { largest possible horizontal extent of pixel image array }
  max_row: integer;  { current vertical extent of pixel image array }
  max_col: integer;  { current horizontal extent of pixel image array }
```

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 

\[(\text{max}_\text{row} + 1) \times (\text{max}_\text{col} + 1)\].

\texttt{METAFONT}'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 \texttt{METAFONT}'s pixel image array \}
\end{verbatim}

\begin{verbatim}
(\text{Constants in the outer block 5*}) \equiv
inf_line_length = 20; sup_line_length = 1023;
\end{verbatim}

This code is used in section 3*.

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 \texttt{bound_var} stuff duplicated in \texttt{tex.ch} \}
define setup_bound_var(#) \equiv bound_default \leftarrow #; setup_bound_var_end
\end{verbatim}

\begin{verbatim}
define setup_bound_var_end(#) \equiv setup_bound_variable(address_of(#), bound_name, bound_default);
\end{verbatim}

\begin{verbatim}
{ Set initial values 6* } \equiv
  \{ See comments in \texttt{tex.ch} for why the name has to be duplicated. \}
  setup_bound_var(\texttt{def_line_length})(´line_length´)(\texttt{line_length});  \{ \texttt{xxx} strings will not produce lines longer than this \}
  setup_bound_var(\texttt{max_image})(´max_rows´)(\texttt{max_rows});  \{ largest allowed vertical extent of pixel image array \}
  setup_bound_var(\texttt{max_image})(´max_cols´)(\texttt{max_cols});  \{ largest allowed horizontal extent of pixel image array \}
  const_chk(\texttt{line_length});
  if max_rows > max_image then max_rows \leftarrow max_image;
  if max_cols > max_image then max_cols \leftarrow max_image;
  \texttt{image_array} \leftarrow \texttt{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 writeLn(stderr,#); uexit(1);
  end
\end{verbatim}

\begin{verbatim}
define bad_gf(#) \equiv abort(´Bad \texttt{GF} file:´,#,´!´)
\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 `GFtype`. So we shall assume that the Pascal system being used for `GFtype` has a character set containing at least the standard visible characters of ASCII code ("!" through "~").

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

\begin{verbatim}
define char ≡ 0..255
define text_char ≡ char \{ 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 = 127 \{ ordinal number of the largest element of text_char \}
\end{verbatim}

(Types in the outer block 8) \textbf{GFtype changes for C } §8

\begin{verbatim}
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.

```c
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;
   \langle Print all the selected options 34* \rangle;
   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 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\}}
\end{verbatim}

\begin{verbatim}
There is no terminal input. The options for running this program are offered through command line options.
\end{verbatim}

\begin{verbatim}
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_in} makes sure that there is always at least one blank space in \texttt{buffer}.

(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.)
\end{verbatim}

\begin{verbatim}
This was so humdrum that we got rid of it. (module 30)
\end{verbatim}

\begin{verbatim}
The dialog procedure module is eliminated. (module 31)
\end{verbatim}

\begin{verbatim}
So is its first part. (module 32)
\end{verbatim}

\begin{verbatim}
So is its second part. (module 33)
\end{verbatim}

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

\begin{verbatim}
\texttt{(Print all the selected options 34*) ≡}
\texttt{print(’Options selected: Mnemonic\_output = ‘);}
\texttt{if \texttt{wants_mnemonics} then print(’true’) else print(’false’);}
\texttt{print(’; Pixel\_output = ‘);}
\texttt{if \texttt{wants_pixels} then print(’true’) else print(’false’);}
\texttt{print_in(’.’)}
\end{verbatim}

This code is used in section 22*. 

This code is used in section 22*. 
In order to allow different systems to change the 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.

\[
\text{define } \text{image} \equiv \text{image} \text{array}[m + (\text{max}\_\text{col} + 1) * n]
\]

(Global in the outer block \text{4*}) +\equiv

image\_\text{array} \uparrow \text{pixel};

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

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

\[
\text{define } \text{max}\_\text{subrow} \equiv \text{max}\_\text{row} \{ \text{vertical size of current subarray of interest} \}
\]

\[
\text{define } \text{max}\_\text{subcol} \equiv \text{max}\_\text{col} \{ \text{horizontal size of current subarray of interest} \}
\]

As we paint the pixels of a character, we will record its actual boundaries in variables \text{max}\_m\_\text{observed} and \text{max}\_n\_\text{observed}. Then the following routine will be called on to output the image, using blanks for white and asterisks for 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.

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

\[
\text{(Print the image 40* ) } \equiv
\begin{align*}
\text{begin} & \{ \text{Compare the subarray boundaries with the observed boundaries 42} \}; \\
\text{if } & \text{max}\_\text{subcol} \geq 0 \text{ then } \{ \text{there was at least one paint command} \}
\end{align*}
\]

\[
\begin{align*}
\text{\{Print asterisk patterns for rows 0 to max}\_\text{subrow 43\}} \\
\text{else print}\_\text{ln} (\text{‘The character is entirely blank.’}); \\
\text{if } (\text{max}\_\text{row} \geq 0) \land (\text{max}\_\text{col} \geq 0) \text{ then} \\
\text{\begin{align*}
\text{begin} & \text{libc}\_\text{free} (\text{image}\_\text{array}); \text{ image}\_\text{array} \leftarrow \text{nil}; \\
\text{\end{align*}} \\
\text{end}; \\
\text{end}
\end{align*}
\]

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

```plaintext
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)]); s ← 10 * (s mod unity); delta ← delta * 10;
    until s ≤ delta;
  end;
end;

48* 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.

```
§51* 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.

\[
\text{(Start translation of command } o \text{ and goto the appropriate label to finish the job 51*) } \equiv \\
\text{ if } o \leq \text{paint1} + 3 \text{ then (Translate a sequence of paint commands, until reaching a non-paint 56);}
\]

\begin{verbatim}
case o of
four_cases(skip0): (Translate a skip command 60);
sixty_four_cases(new_row_0): (Translate a new_row command 59);
sixty_four_cases(new_row_0 + 64): (Translate a new_row command 59);
thirty_seven_cases(new_row_0 + 128): (Translate a new_row command 59);
\end{verbatim}

(Cases for commands no_op, pre, post, postpost, boc, and eoc 52)

\begin{verbatim}
four_cases(xxx1): (Translate an xxx command 53);
gyy: (Translate a gyy command 55);
othercases error(`undefined command`, `o : 1`, `!`)
endcases
\end{verbatim}

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

\begin{verbatim}
begin initialize; \{ get all variables initialized \}
\langle Process the preamble 68 \rangle;
\langle Translate all the characters 69 \rangle;
print\_nl; read\_postamble; print\(\langle \text{The file had} \rangle, total\_chars : 1, \langle \text{character} \rangle); 
if total\_chars \neq 1 then print\(\langle s \rangle); 
print\_ln(\langle altogether. \rangle); 
end.
\end{verbatim}
§73  **System-dependent changes.**  Parse a Unix-style command line.

    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;
        getopt_return_val: integer;  option_index: c_int_type;  current_option: 0 .. n_options;
    begin  {Define the option table 74*};
        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(GFTYPE_HELP, nil);
            end
            else if argument_is(‘version’) then
                begin print_version_and_exit(banner, nil, ‘D.R. Fuchs’, nil);
            end
        until getopt_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 write_in(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.

(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.

(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?

(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);
Show pixels?

(Define the option table 74*) \[ \equiv \]
\[
\text{long_options}[\text{current_option}].\text{name} \leftarrow \text{images}; \text{long_options}[\text{current_option}].\text{has_arg} \leftarrow 0; \\
\text{long_options}[\text{current_option}].\text{flag} \leftarrow \text{address_of}(\text{wants_pixels}); \text{long_options}[\text{current_option}].\text{val} \leftarrow 1; \\
\text{incr}(\text{current_option});
\]

An element with all zeros always ends the list.

(Define the option table 74*) \[ \equiv \]
\[
\text{long_options}[\text{current_option}].\text{name} \leftarrow 0; \text{long_options}[\text{current_option}].\text{has_arg} \leftarrow 0; \\
\text{long_options}[\text{current_option}].\text{flag} \leftarrow 0; \text{long_options}[\text{current_option}].\text{val} \leftarrow 0;
\]
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.

-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.
dx: 15, 18.
dy: 15, 18.
eight_bits: 20, 24, 48* 49.
eight_cases: 48*.
else: 2.
end: 2.
endcases: 2.
eoc: 13, 15, 16, 17, 18, 38* 42, 44, 48* 49, 52, 69, 71.

b: 24, 67.
backpointer...should be p: 61.
back pointers: 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.

const_char: 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.
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.
othercases: 2.
others: 2.
output: 3*
p: 49, 61, 67.
paint: 56.
paint_switch: 14, 15, 35, 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_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_ln: 3* 43, 50, 52, 53, 66*, 69.
print_scaled: 45*, 55, 61, 65.
print_version_and_exit: 73*
proofing: 18.
q: 49, 61, 67.
r: 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_end: 6*
ssetup_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*.  
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*.  
three_seven_cases: 48* 51*  
three_two_cases: 48*  
This pixel's lower...: 43.  
This pixel's upper: 43.  
total_chars: 46, 47, 66*, 71.  
true: 25*, 36, 49, 53.  
w: 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*.  
vppp: 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*, 4*, 73*.  
xcalloc_array: 38*.  
xchr: 10, 11, 12, 45*, 53, 68.  
xord: 10, 12.  
xxx1: 15, 16, 48*, 51*, 70.
\(
\) Names of the Sections

GF type changes for C

(Cases for commands \texttt{no\_op}, \texttt{pre}, \texttt{post}, \texttt{post\_post}, \texttt{boc}, and \texttt{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*.

(Constats in the outer block 5* ) Used in section 3*.

(Define the option table \texttt{74*}, \texttt{75*}, \texttt{76*}, \texttt{77*}, \texttt{78*} ) Used in section 73*.

(Define \texttt{parse\_arguments} 73* ) Used in section 3*.


(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 \texttt{boc} command 71) Used in section 69.

(Pass an \texttt{eoc} command 72) Used in section 69.

(Pass \texttt{no\_op}, \texttt{xxx} and \texttt{yyy} commands 70) Used in section 69.

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

(Print asterisk patterns for rows 0 to \texttt{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 \texttt{o} and \texttt{goto} the appropriate label to finish the job 51*) Used in section 50.

(Translate a sequence of \texttt{paint} commands, until reaching a non-\texttt{paint} 56) Used in section 51*.

(Translate a \texttt{new\_row} command 59) Used in sections 51*, 51*, and 51*.

(Translate a \texttt{skip} command 60) Used in section 51*.

(Translate a \texttt{yyy} command 55) Used in sections 51* and 70.

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

(Translate an \texttt{xxx} command 53) Used in sections 51* and 70.

(Translate the next command in the \texttt{GF} file; \texttt{goto} 9999 if it was \texttt{eoc}; \texttt{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*.