The PKtoGF processor

(Version 1.1, 22 April 2020)
2* The banner string defined here should be changed whenever PKtoGF gets modified. You should update the preamble comment as well.

```plaintext
define my_name ≡ 'pktogf'
define banner ≡ 'This_is_PKtoGF_Version_1.1' { printed when the program starts }
define preamble_comment ≡ 'PKtoGF_1.1_output'
define comm_length ≡ 17
```

4* Both the input and output come from binary files. On line interaction is handled through Pascal’s standard input and output files. For C compilation terminal input and output is directed to stdin and stdout. In this program there is no terminal input. Since the terminal output is really not very interesting, it is produced only when the −v command line flag is presented.

```plaintext
define print_in(#) ≡ 
  if verbose then write_in(output,#)
define print(#) ≡ 
  if verbose then write(output,#)
program PKtoGF(input, output);
const ⟨Constants in the outer block 6*⟩
type ⟨Types in the outer block 9⟩
var ⟨Globals in the outer block 11⟩
  ⟨Define parse_arguments 74*⟩
procedure initialize; { this procedure gets things started properly }
  var i: integer; { loop index for initializations }
  begin kpse_set_program_name(argv[0], my_name); kpse_init_prog("PKTOGF", 0, nil, nil);
    parse_arguments; print_in(banner);
    ⟨Set initial values 12⟩
  end;
```

5* This module is deleted, because it is only useful for a non-local goto, which we don’t use in C.

6* These constants determine the maximum length of a file name and the length of the terminal line, as well as the maximum number of run counts allowed per line of the GF file. (We need this to implement repeat counts.)

```plaintext
  ⟨Constants in the outer block 6*⟩ ≡ 
  MAX_Counts = 400;  { initial number of run counts in a raster line }
```

This code is used in section 4*.

8* It is possible that a malformed packed file (heaven forbid!) or some other error might be detected by this program. Such errors might occur in a deeply nested procedure, so we might want to abort the program with an error message.

```plaintext
define abort(#) ≡ 
  begin verbose ← true; print_in(#); uexit(1);
  end
```
10* 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 GFD to PK. So we shall assume that the Pascal system being used for GFD to PK 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  { \texttt{text.char} }
define first_text_char = 0  { ordinal number of the smallest element of \texttt{text.char} }
define last_text_char = 127  { ordinal number of the largest element of \texttt{text.char} }
\end{verbatim}

(\text{Types in the outer block 9}) +≡

\begin{verbatim}
text_file = packed file of text_char;
\end{verbatim}
The final algorithm for decoding the run counts based on the above scheme might look like this, assuming a procedure called `pk_nyb` is available to get the next nybble from the file, and assuming that the global `repeat_count` indicates whether a row needs to be repeated. Note that this routine is recursive, but since a repeat count can never directly follow another repeat count, it can only be recursive to one level.

(Packed number procedure 30*) ≡

```plaintext
function pk_packed_num: integer;
  var i, j: integer;
  begin i ← get_nyb;
  if i = 0 then
    begin repeat j ← get_nyb; incr(i);
    until j ≠ 0;
    while i > 0 do
      begin j ← j * 16 + get_nyb; decr(i);
      end;
      pk_packed_num ← j - 15 + (13 - dyn_f) * 16 + dyn_f;
    end
  else if i ≤ dyn_f then pk_packed_num ← i
  else if i < 14 then pk_packed_num ← (i - dyn_f - 1) * 16 + get_nyb + dyn_f + 1
  else begin if i = 14 then repeat_count ← pk_packed_num
    else repeat_count ← 1;
    pk_packed_num ← pk_packed_num;
    end;
  end;
  This code is used in section 62.
```
To prepare these files for input, we reset them. An extension of Pascal is needed in the case of gf_file, since we want to associate it with external files whose names are specified dynamically (i.e., not known at compile time). The following code assumes that ‘reset(f, s)’ does this, when f is a file variable and s is a string variable that specifies the file name. If eof(f) is true immediately after reset(f, s) has acted, we assume that no file named s is accessible.

In C, we do path searching based on the user’s environment or the default path, via the Kpathsea library.

procedure open_pk_file;  { prepares to read packed bytes in pk_file }
begin
  { Don’t use kpse_find_pk; we want the exact file or nothing. }
  pk_name ← cmdline(optind); pk_file ← kpse_open_file(cmdline(optind), kpse_pk_format);
  if pk_file then
    begin cur_loc ← 0;
    end;
end;

procedure open_gf_file;  { prepares to write packed bytes in gf_file }
begin
  { If an explicit output filename isn’t given, we construct it from pk_name. }
  if optind + 1 = argc then
    begin gf_name ← basename_change_suffix(pk_name, 'pk', 'gf');
    end
  else begin gf_name ← cmdline(optind + 1);
    end;
  rewritebin(gf_file, gf_name); gf_loc ← 0;
end;

No arbitrary limit on filename length.

Globals in the outer block 11
≡
gf_name, pk_name: c_string;  { names of input and output files }
gf_loc, pk_loc: integer;  { how many bytes have we sent? }

Byte output is handled by a C definition.

define gf_byte(#) ≡
  begin put_byte(#, gf_file); incr(gf_loc)
  end
43* We shall use a set of simple functions to read the next byte or bytes from \textit{pk\_file}. There are seven possibilities, each of which is treated as a separate function in order to minimize the overhead for subroutine calls.

\begin{verbatim}
define pk\_byte \equiv get\_byte
define pk\_loc \equiv cur\_loc

function get\_byte: integer; \{ returns the next byte, unsigned \}
var b: eight\_bits;
begin if eof (pk\_file) then get\_byte \leftarrow 0
else begin read(pk\_file, b); incr(cur\_loc); get\_byte \leftarrow b;
end;
end;

function signed\_byte: integer; \{ returns the next byte, signed \}
var b: eight\_bits;
begin read(pk\_file, b); incr(cur\_loc);
if b < 128 then signed\_byte \leftarrow b else signed\_byte \leftarrow b - 256;
end;

function get\_two\_bytes: integer; \{ returns the next two bytes, unsigned \}
var a, b: eight\_bits;
begin read(pk\_file, a); read(pk\_file, b); cur\_loc \leftarrow cur\_loc + 2; get\_two\_bytes \leftarrow a \ast 256 + b;
end;

function signed\_pair: integer; \{ returns the next two bytes, signed \}
var a, b: eight\_bits;
begin read(pk\_file, a); read(pk\_file, b); cur\_loc \leftarrow cur\_loc + 2;
if a < 128 then signed\_pair \leftarrow a \ast 256 + b
else signed\_pair \leftarrow (a - 256) \ast 256 + b;
end;

function get\_three\_bytes: integer; \{ returns the next three bytes, unsigned \}
var a, b, c: eight\_bits;
begin read(pk\_file, a); read(pk\_file, b); read(pk\_file, c); cur\_loc \leftarrow cur\_loc + 3;
get\_three\_bytes \leftarrow (a \ast 256 + b) \ast 256 + c;
end;
@

function signed\_trio: integer; \{ returns the next three bytes, signed \}
var a, b, c: eight\_bits;
begin read(pk\_file, a); read(pk\_file, b); read(pk\_file, c); cur\_loc \leftarrow cur\_loc + 3;
if a < 128 then signed\_trio \leftarrow (a \ast 256 + b) \ast 256 + c
else signed\_trio \leftarrow ((a - 256) \ast 256 + b) \ast 256 + c;
end;
@

function signed\_quad: integer; \{ returns the next four bytes, signed \}
var a, b, c, d: eight\_bits;
begin read(pk\_file, a); read(pk\_file, b); read(pk\_file, c); read(pk\_file, d); cur\_loc \leftarrow cur\_loc + 4;
if a < 128 then signed\_quad \leftarrow ((a \ast 256 + b) \ast 256 + c) \ast 256 + d
else signed\_quad \leftarrow (((a - 256) \ast 256 + b) \ast 256 + c) \ast 256 + d;
end;
\end{verbatim}
§45  We put definitions here to access the DVItype functions supplied above. \((\text{signed}_\text{byte} \text{ is already taken care of})\).

\[
\begin{align*}
\text{define } & \text{get}_\text{16} \equiv \text{get}_\text{two}_\text{bytes} \\
\text{define } & \text{signed}_\text{16} \equiv \text{signed}_\text{pair} \\
\text{define } & \text{get}_\text{32} \equiv \text{signed}_\text{quad}
\end{align*}
\]

46* As we are writing the GF file, we often need to write signed and unsigned, one, two, three, and four-byte values. These routines give us that capability.

\[
\begin{align*}
\text{procedure } & \text{gf}_\text{16}(i : \text{integer}); \\
& \text{begin } \text{gf}_\text{byte}(i \div 256); \text{gf}_\text{byte}(i \mod 256); \\
& \text{end};
\end{align*}
\]

\[
\begin{align*}
\text{procedure } & \text{gf}_\text{24}(i : \text{integer}); \\
& \text{begin } \text{gf}_\text{byte}(i \div 65536); \text{gf}_\text{16}(i \mod 65536); \\
& \text{end};
\end{align*}
\]

\[
\begin{align*}
\text{procedure } & \text{gf}_\text{quad}(i : \text{integer}); \\
& \text{begin } \text{if } i \geq 0 \text{ then } \\
& \quad \text{begin } \text{gf}_\text{byte}(i \div 16777216); \\
& \quad \text{end } \\
& \text{else begin } \{ i < 0 \text{ at this point, but a compiler is permitted to rearrange the order of the additions, which would cause wrong results in the unlikely event of a non-2’s-complement representation. } \} \\
& \quad i \leftarrow i + 1073741824; i \leftarrow i + 1073741824; \text{gf}_\text{byte}(128 + (i \div 16777216)); \\
& \quad \text{end}; \\
& \text{gf}_\text{24}(i \mod 16777216); \\
& \text{end};
\end{align*}
\]
Now we read and check the preamble of the PK file. In the preamble, we find the \texttt{hppp}, \texttt{design\_size}, \texttt{checksum}. We write the relevant parameters to the GF file, including the preamble comment.

\begin{verbatim}
(Read preamble 49*) ≡
if pk\_byte ≠ pk\_pre then abort(‘Bad\_pk\_file!\_pre\_command\_missing.’);
gf\_byte(pre);
if pk\_byte ≠ pk\_id then abort(‘Wrong\_version\_of\_packed\_file!’);
gf\_byte(gf\_id\_byte); j ← pk\_byte; gf\_byte(j); print(‘{’);
for i ← 1 to j do
  begin hppp ← pk\_byte; gf\_byte(hppp); print(xchr[xord[hppp]])
  end;
print\_ln(‘}’); design\_size ← get\_32; checksum ← get\_32; hppp ← get\_32; vppp ← get\_32;
if hppp ≠ vppp then print\_ln(‘Warning:\_aspect\_ratio\_not\_1:1!’);
magnification ← round(hppp * 72.27 * 5/65536); last\_eoc ← gf\_loc
\end{verbatim}

This code is used in section 73*.

\begin{verbatim}
(Set initial values 12 ) ≡
  row\_counts ← xmalloc\_array(integer, MAX\_COUNTS); max\_counts ← MAX\_COUNTS;
\end{verbatim}

Now, the globals to help communication between these procedures, and a buffer for the raster row counts.

\begin{verbatim}
(Globals in the outer block 11 ) ≡
input\_byte: eight\_bits; \{ the byte we are currently decimating \}
bit\_weight: eight\_bits; \{ weight of the current bit \}
max\_counts: integer;
row\_counts: ↑integer; \{ where the row is constructed \}
rcp: integer; \{ the row counts pointer \}
\end{verbatim}
65* And the main procedure.

\begin{verbatim}
\langle Read and translate raster description 65* \rangle \equiv
\end{verbatim}

\begin{verbatim}
begin
  if \((c_{\text{width}} > 0) \land (c_{\text{height}} > 0)\) then
    begin
      bit_weight \leftarrow 0; count_down \leftarrow c_{\text{height}} \ast c_{\text{width}} - 1;
      if dyn_f = 14 then turn_on \leftarrow \text{get bit};
      repeat_count \leftarrow 0; x_{\text{to go}} \leftarrow c_{\text{width}}; y_{\text{to go}} \leftarrow c_{\text{height}};
      cur_n \leftarrow c_{\text{height}}; count \leftarrow 0;
      first_on \leftarrow turn_on; turn_on \leftarrow \neg turn_on; rcp \leftarrow 0;
      while \(y_{\text{to go}} > 0\) do
        begin
          if count = 0 then \langle Get next count value into count 64 \rangle;
          if rcp = 0 then first_on \leftarrow turn_on;
          while count \geq x_{\text{to go}} do
            begin
              row_counts[rcp] \leftarrow x_{\text{to go}}; count \leftarrow count - x_{\text{to go}};
              for i \leftarrow 0 \text{ to repeat_count} do
                begin
                  (Output row 66);
                  y_{\text{to go}} \leftarrow y_{\text{to go}} - 1;
                end;
            end;
          if (count > 0) then first_on \leftarrow turn_on;
        end;
        if count > 0 then
          begin
            row_counts[rcp] \leftarrow count;
            if rcp = 0 then first_on \leftarrow turn_on;
            rcp \leftarrow rcp + 1;
            if rcp > max_counts then
              begin
                printLn( \langle Reallocated row_counts array to \rangle, (max_counts + MAX_COUNTS) : 1,
                \langle \langle items \rangle from, max_counts : 1, \rangle : \rangle; max_counts \leftarrow max_counts + MAX_COUNTS;
                row_counts \leftarrow xrealloc_array(row_counts, integer, max_counts);
              end;
          end;
        end;
      x_{\text{to go}} \leftarrow x_{\text{to go}} - count; count \leftarrow 0;
    end;
end
\end{verbatim}

This code is used in section 47.
71* Terminal communication. Since this program runs entirely on command-line arguments, there is no terminal communication.

72* pktogf.web has a *dialog* procedure here.
§73. **The main program.** Now that we have all the pieces written, let us put them together.

```plaintext
begin initialize; ⟨Open files 44⟩;
⟨Read preamble 49⟩;
skip_specials;
while flag_byte ≠ pk_post do
  begin ⟨Unpack and write character 47⟩;
skip_specials;
  end;
while ¬eof (pk_file) do i ← pk_byte;
⟨Write GF postamble 68⟩;
print_ln (pk_loc : 1, "bytes unpacked", gf_loc : 1, "bytes.");
end.
```
74* System-dependent changes. Parse a Unix-style command line.

```
define argument_is(#) ≡ (strcmp(long_options[option_index].name, #) = 0)
  (Define parse arguments 74*)

procedure parse_arguments;
  const n_options = 3; {Pascal won’t count array lengths for us.}
  var long_options: array [0..n_options] of getopt_struct;
  getopt_return_val: integer; option_index: c_int_type; current_option: 0..n_options;
  begin (Initialize the option variables 79*);
    (Define the option table 75*);
    repeat getopt_return_val ← getopt_long_only(argc, argv, *, long_options, address_of(current_option));
      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(PKTOGF_HELP, nil);
        end
      else if argument_is(’version’) then
        begin print_version_and_exit(banner, nil, ’Tomas.Rokicki’, nil);
        end{ Else it was a flag; getopt has already done the assignment.}
  until getopt_return_val = −1; {Now optind is the index of first non-option on the command line. We
    must have one or two remaining arguments.}
  if (optind + 1 ≠ argc) ∧ (optind + 2 ≠ argc) then
    begin write_line(stderr, my_name,’:\$Need one or two file arguments.’); usage(my_name);
    end;
end;
```

This code is used in section 4*.

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

```
(Define the option table 75*)
  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 76*, 77*, and 80*.

This code is used in section 74*.

76* Another of the standard options.

```
(Define the option table 75*)
  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);
```

77* Print progress information?

```
(Define the option table 75*)
  long_options[current_option].name ← ’verbose’; long_options[current_option].has_arg ← 0;
  long_options[current_option].flag ← address_of(verbos);
  long_options[current_option].val ← 1; incr(current_option);
```

78* (Globals in the outer block 11) +=

```
verbose: c_int_type;
```
§79  PK to GF changes for C

(Initialize the option variables 79*) \equiv

\textit{verbose} \leftarrow \textit{false};

This code is used in section 74*.

80*  An element with all zeros always ends the list.

(Define the option table 75*) +≡

\textit{long_options}[\textit{current_option}].\textit{name} \leftarrow 0; \textit{long_options}[\textit{current_option}].\textit{has_arg} \leftarrow 0;
\textit{long_options}[\textit{current_option}].\textit{flag} \leftarrow 0; \textit{long_options}[\textit{current_option}].\textit{val} \leftarrow 0;
81* 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: 2, 4, 5, 6, 8, 10, 30, 40, 41, 42, 43, 45, 46, 49, 51, 63, 65, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81.

- help: 75*
- version: 76*
    a: 43*
    abort: 8* 47, 49*, 70.
    address_of: 74*, 77*
    argc: 40*, 74*
    argument_is: 74*
    argv: 4*, 74*
    ASCII_code: 9, 11.
    b: 43*
    backpointers: 19.
    banner: 2*, 4*, 74*
    basename_change_suffix: 40*
    bit_weight: 62, 63*, 65*
    black: 15, 16.
    boc: 14, 16, 17, 18, 19, 59.
    boc1: 16, 17, 59.
    byte_file: 38, 39.
    c: 43*
    c_height: 52, 53, 54, 55, 56, 65*
    c_int_type: 74*, 78*
    c_string: 41*
    c_width: 52, 53, 54, 55, 56, 65*
    car: 48, 52, 53, 54, 59, 60.
    cc: 32.
    char: 10*, 50.
    char_loc: 16, 17, 19, 61.
    char_loc0: 16, 17, 61.
    char_pointer: 57, 58, 59, 60, 61.
    check_sum: 18.
    checksum: 49*, 50, 68.
    Chinese characters: 19.
    chr: 10*, 11, 13.
    cmdline: 40*
    comm_length: 2*, 50.
    comment: 50.
    count: 64, 65*, 67.
    count_down: 64, 65*, 67.
    cs: 18, 23.
    cur_loc: 40*, 43*
    current_option: 74*, 75*, 76*, 77*, 80*
    d: 43*
    decr: 7, 30*
    del_m: 16.
    del_n: 16.
    design size: 18.
    design_size: 49*, 50, 68.
    dialog: 72*
    dm: 16, 32.
    do_nothing: 7, 74*
    done: 64, 67.
    ds: 18, 23.
    dx: 16, 19, 32.
    dy: 16, 19, 32.
    dyn_f: 28, 29, 30*, 31, 32, 35, 36, 47, 48, 64, 65*
    eight_bits: 38, 43*, 62, 63*
    else: 3.
    end: 3.
    endcases: 3.
    coc: 14, 16, 17, 18, 47.
    cof: 40*, 43*, 73*
    false: 64, 79*
    flag: 32, 75*, 76*, 77*, 80*
    flag_byte: 47, 53, 54, 69, 70, 73*
    Fuchs, David Raymond: 20.
    get_bit: 62, 64, 65*
    get_byte: 43*
    get_nb: 30*, 62.
    get_three_bytes: 43*
    get_two_bytes: 43*, 45*
    get16: 45*, 53, 54.
    get32: 45*, 49*, 52, 70.
    getopt: 74*
    getopt_long_only: 74*
    getopt_return_val: 74*
    getopt_struct: 74*
    gf_byte: 38, 42*, 46*, 47, 49*, 59, 61, 66, 68, 70.
    gf_file: 39, 40*, 42*
    gf_id_byte: 16, 49*, 68.
    gf_loc: 40*, 41*, 42*, 47, 49*, 68, 70, 73*
    gf_name: 40*, 41*
    gf_quad: 46*, 59, 61, 68, 70.
    gf16: 46*, 66.
    gf24: 46*
    has_arg: 75*, 76*, 77*, 80*
    height: 31.
    hoff: 32, 34.
    hor_esc: 52, 53, 54, 55, 60.
    hppp: 18, 23, 49*, 50, 68.
    i: 4*, 30*, 48, 70.
    incr: 7, 30*, 42*, 43*, 75*, 76*, 77*.
initialize: 4* 73*
input: 4*
input_byte: 62, 63*
j: 48.
kps_find_pk: 40*
kps_init_prog: 4*
kps_open_file: 40*
kps_pk_format: 40*
kps_set_program_name: 4*
lst_eoc: 47, 49*, 55, 68.
lst_text_char: 10*, 13.
long_options: 74*, 75*, 76*, 77*, 80*.
magnification: 49*, 50.
max: 66, 67.
max_counts: 51*, 63*, 65*.
MAX_COUNTS: 6*, 51*, 65*.
max_m: 16, 18, 56, 57, 59.
max_n: 16, 18, 56, 57, 59.
max_new_row: 17.
min_m: 16, 18, 56, 57, 59.
min_n: 16, 18, 56, 57, 59.
mmax_m: 56, 57, 58, 68.
mmax_n: 56, 57, 58, 68.
mmin_m: 56, 57, 58, 68.
mmin_n: 56, 57, 58, 68.
my_name: 2*, 4* 74*.
no_options: 74*.
nname: 74*, 75*, 76*, 77*, 80*.
new_row_0: 16, 17, 66.
new_row_1: 16.
new_row_164: 16.
no_op: 16, 17, 19.
nop: 17.
opend_file: 40*, 44.
opend_pk_file: 40*, 44.
opind: 40*, 74*.
option_index: 74*.
ord: 11.
oriental characters: 19.
othercases: 3.
others: 3.
output: 4*.
packet_length: 52, 53, 54, 55.
paint_switch: 15, 16.
paint_0: 16, 17, 66.
paint1: 16, 17, 66.
paint2: 16.
paint3: 16.
parse_arguments: 4* 74*.
pl: 32.
plot: 14, 16, 17, 18, 20, 68.
post: 14, 16, 17, 18, 20, 68.
pre: 14, 16, 17, 49*.
preamble_comment: 2*.
print: 4* 49*.
print_line: 4*, 8*, 49*, 60, 65*, 73*.
print_version_and_exit: 74*.
proofing: 19.
put_byte: 42*.
rtp: 63*, 65*, 66.
read: 43*.
repeat_count: 30*, 65*, 67*.
reset: 40*.
rewritebin: 40*.
round: 49*.
row_counts: 51*, 63*, 65*, 66*.
s_hor_esc: 57, 60, 61.
s_tfm_width: 57, 60, 61.
s_ver_esc: 57, 60, 61.
scaled: 16, 18, 19, 23.
signed_byte: 43*, 45*, 54*.
signed_pair: 43*, 45*.
signed_quad: 43*, 45*.
signed_trio: 43*.
signed_16: 45*, 53*.
skip_specials: 70, 73*.
skip0: 16, 17, 66.
skip1: 16, 17, 66.
skip2: 16.
skip3: 16.
stderr: 74*.
stdin: 4*.
stdout: 4*.
strcmp: 74*.
system dependencies: 6*, 38*.
system dependencies: 10*, 20, 39, 40*, 43*.
temp: 62.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>text_char</td>
<td>(10^*) 11.</td>
</tr>
<tr>
<td>text_file</td>
<td>(10^*)</td>
</tr>
<tr>
<td>tfm</td>
<td>32, 33, 36.</td>
</tr>
<tr>
<td>tfm_width</td>
<td>48, 52, 53, 54, 60.</td>
</tr>
<tr>
<td>this_char_ptr</td>
<td>57, 59, 70.</td>
</tr>
<tr>
<td>true</td>
<td>(8^*) 64.</td>
</tr>
<tr>
<td>turn_on</td>
<td>47, 64, 65*, 66, 67.</td>
</tr>
<tr>
<td>uexit</td>
<td>(8^*)</td>
</tr>
<tr>
<td>undefined_commands</td>
<td>17.</td>
</tr>
<tr>
<td>usage</td>
<td>(74^*)</td>
</tr>
<tr>
<td>usage_help</td>
<td>(74^*)</td>
</tr>
<tr>
<td>val</td>
<td>(75^<em>, 76^</em>, 77^<em>, 80^</em>)</td>
</tr>
<tr>
<td>ver_esc</td>
<td>52, 53, 54, 55, 60.</td>
</tr>
<tr>
<td>verbose</td>
<td>(4^<em>) (8^</em>, 77^<em>, 78^</em>, 79^*)</td>
</tr>
<tr>
<td>voff</td>
<td>32, 34.</td>
</tr>
<tr>
<td>uppp</td>
<td>18, 23, 49*, 50, 68.</td>
</tr>
<tr>
<td>white</td>
<td>16.</td>
</tr>
<tr>
<td>width</td>
<td>31.</td>
</tr>
<tr>
<td>word_width</td>
<td>52, 53, 54, 55.</td>
</tr>
<tr>
<td>write</td>
<td>(4^*)</td>
</tr>
<tr>
<td>write_ln</td>
<td>(4^<em>) (74^</em>)</td>
</tr>
<tr>
<td>x_off</td>
<td>48, 52, 53, 54, 56.</td>
</tr>
<tr>
<td>x_to_go</td>
<td>65*, 67.</td>
</tr>
<tr>
<td>xchr</td>
<td>11, 12, 13, 49*</td>
</tr>
<tr>
<td>xmalloc_array</td>
<td>51*</td>
</tr>
<tr>
<td>xord</td>
<td>11, 13, 49*</td>
</tr>
<tr>
<td>xrealloc_array</td>
<td>65*</td>
</tr>
<tr>
<td>xxx1</td>
<td>16, 17.</td>
</tr>
<tr>
<td>xxx2</td>
<td>16.</td>
</tr>
<tr>
<td>xxx3</td>
<td>16.</td>
</tr>
<tr>
<td>xxx4</td>
<td>16.</td>
</tr>
<tr>
<td>y_off</td>
<td>48, 52, 53, 54, 56.</td>
</tr>
<tr>
<td>y_to_go</td>
<td>65*, 66, 67.</td>
</tr>
<tr>
<td>yyy</td>
<td>16, 17, 19, 23.</td>
</tr>
</tbody>
</table>
Calculate and check \( \min_m, \max_m, \min_n, \text{ and } \max_n \) \( \text{used in section 47.} \)

Constants in the outer block \( 6^* \) \( \text{used in section 4*.} \)

Define the option table \( 75^*, 76^*, 77^*, 80^* \) \( \text{used in section 74*.} \)

Define \( \text{parse_arguments 74*} \) \( \text{used in section 4*.} \)

Get next count value into \( \text{count 64} \) \( \text{used in section 65*.} \)

Globals in the outer block \( 11, 39, 41^*, 48, 50, 55, 57, 63^*, 67, 69, 78^* \) \( \text{used in section 4*.} \)

Initialize the option variables \( 79^* \) \( \text{used in section 74*.} \)

Open files \( 44 \) \( \text{used in section 73*.} \)

Output row \( 66 \) \( \text{used in section 65*.} \)

Packed number procedure \( 30^* \) \( \text{used in section 62.} \)

Read and translate raster description \( 65^* \) \( \text{used in section 47.} \)

Read extended short character preamble \( 53 \) \( \text{used in section 47.} \)

Read long character preamble \( 52 \) \( \text{used in section 47.} \)

Read preamble \( 49^* \) \( \text{used in section 73*.} \)

Read short character preamble \( 54 \) \( \text{used in section 47.} \)

Save character locator \( 60 \) \( \text{used in section 47.} \)

Set initial values \( 12, 13, 51^*, 58 \) \( \text{used in section 4*.} \)

Types in the outer block \( 9, 10^*, 38 \) \( \text{used in section 4*.} \)

Unpack and write character \( 47 \) \( \text{used in section 73*.} \)

Write GF postamble \( 68 \) \( \text{used in section 73*.} \)

Write character locators \( 61 \) \( \text{used in section 68.} \)

Write character preamble \( 59 \) \( \text{used in section 47.} \)