

The PKtoGF processor

(Version 1.1, 22 April 2020)

	Section	Page
Introduction	1	2
The character set	9	3
Generic font file format	14	4
Packed file format	21	4
Input and output	38	5
Character unpacking	47	8
Terminal communication	71	10
The main program	73	11
System-dependent changes	74	12
Index	81	14

Editor's Note: The present variant of this C/WEB source file has been modified for use in the T_EX Live system.

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](#).

The preparation of this report was supported in part by the National Science Foundation under grants IST-8201926 and MCS-8300984, and by the System Development Foundation. 'T_EX' is a trademark of the American Mathematical Society.

2* The *banner* string defined here should be changed whenever **PKtoGF** gets modified. You should update the preamble comment as well.

```

define my_name  $\equiv$  `pktogf`
define banner  $\equiv$  `This_is_PKtoGF,_Version_1.1` { printed when the program starts }
define preamble_comment  $\equiv$  `PKtoGF_1.1_output`
define comm_length  $\equiv$  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.

```

define print_ln(#)  $\equiv$ 
    if verbose then write_ln(output, #)
define print(#)  $\equiv$ 
    if verbose then write(output, #)
program PKtoGF(input, output);
const  $\langle$  Constants in the outer block 6*  $\rangle$ 
type  $\langle$  Types in the outer block 9  $\rangle$ 
var  $\langle$  Globals in the outer block 11  $\rangle$ 
     $\langle$  Define parse_arguments 74*  $\rangle$ 
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_ln(banner);
     $\langle$  Set initial values 12  $\rangle$ 
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.)

```

 $\langle$  Constants in the outer block 6*  $\rangle \equiv$ 
    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.

```

define abort(#)  $\equiv$ 
    begin verbose  $\leftarrow$  true; print_ln(#); uxit(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 **GFtoPK**. So we shall assume that the Pascal system being used for **GFtoPK** 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 *chr(first_text_char)* through *chr(last_text_char)*, inclusive. The following definitions should be adjusted if necessary.

define *char* \equiv 0 .. 255

define *text_char* \equiv *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* }

⟨ Types in the outer block 9 ⟩ +≡

text_file = **packed file of** *text_char*;

30* 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*⟩ ≡

```

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.

40* 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;
```

41* 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? }

42* 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 *pk_file*. There are seven possibilities, each of which is treated as a separate function in order to minimize the overhead for subroutine calls.

```

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 * 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 * 256 + b
  else signed_pair  $\leftarrow$  (a - 256) * 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 * 256 + b) * 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 * 256 + b) * 256 + c
  else signed_trio  $\leftarrow$  ((a - 256) * 256 + b) * 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 * 256 + b) * 256 + c) * 256 + d
  else signed_quad  $\leftarrow$  (((a - 256) * 256 + b) * 256 + c) * 256 + d;
  end;

```

45* We put definitions here to access the `DVIttype` functions supplied above. (*signed_byte* is already taken care of).

```
define get_16  $\equiv$  get_two_bytes
define signed_16  $\equiv$  signed_pair
define get_32  $\equiv$  signed_quad
```

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.

```
procedure gf_16(i : integer);
begin gf_byte(i div 256); gf_byte(i mod 256);
end;

procedure gf_24(i : integer);
begin gf_byte(i div 65536); gf_16(i mod 65536);
end;

procedure gf_quad(i : integer);
begin if i  $\geq$  0 then
  begin gf_byte(i div 16777216);
  end
else begin { i < 0 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. }
  i  $\leftarrow$  i + 1073741824; i  $\leftarrow$  i + 1073741824; gf_byte(128 + (i div 16777216));
  end;
gf_24(i mod 16777216);
end;
```

49* Now we read and check the preamble of the PK file. In the preamble, we find the *hppp*, *design_size*, *checksum*. We write the relevant parameters to the GF file, including the preamble comment.

```

⟨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

```

This code is used in section 73*.

51* ⟨Set initial values 12⟩ +≡
row_counts ← *xmalloc_array(integer, MAX_COUNTS)*; *max_counts* ← *MAX_COUNTS*;

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

```

⟨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 }

```


65* And the main procedure.

⟨Read and translate raster description 65*⟩ ≡

```

if (c_width > 0) ∧ (c_height > 0) then
  begin bit_weight ← 0; count_down ← c_height * c_width - 1;
  if dyn_f = 14 then turn_on ← get_bit;
  repeat_count ← 0; x_to_go ← c_width; y_to_go ← c_height; cur_n ← c_height; count ← 0;
  first_on ← turn_on; turn_on ← ¬turn_on; rcp ← 0;
  while y_to_go > 0 do
    begin if count = 0 then ⟨Get next count value into count 64⟩;
    if rcp = 0 then first_on ← turn_on;
    while count ≥ x_to_go do
      begin row_counts[rcp] ← x_to_go; count ← count - x_to_go;
      for i ← 0 to repeat_count do
        begin ⟨Output row 66⟩;
        y_to_go ← y_to_go - 1;
        end;
      repeat_count ← 0; x_to_go ← c_width; rcp ← 0;
      if (count > 0) then first_on ← turn_on;
      end;
    if count > 0 then
      begin row_counts[rcp] ← count;
      if rcp = 0 then first_on ← turn_on;
      rcp ← rcp + 1;
      if rcp > max_counts then
        begin print_ln(‘Reallocated_row_counts_array_to_’, (max_counts + MAX_COUNTS) : 1,
          ‘_items_from_’, max_counts : 1, ‘.’); max_counts ← max_counts + MAX_COUNTS;
        row_counts ← xrealloc_array(row_counts, integer, max_counts);
        end;
      x_to_go ← x_to_go - count; count ← 0;
      end;
    end;
  end

```

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.

```

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_to_´, gf_loc : 1, ´_bytes.´);
end.

```

74* **System-dependent changes.** Parse a Unix-style command line.

```

define argument_is(#)  $\equiv$  (strcmp(long_options[option_index].name, #) = 0)
⟨Define parse_arguments 74*⟩  $\equiv$ 
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  $\leftarrow$  getopt_long_only(argc, argv,  $\text{'\textasciitilde'}$ , 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( $\text{'help'}$ ) then
      begin usage_help(PKTOGF_HELP, nil);
      end
    else if argument_is( $\text{'version'}$ ) then
      begin print_version_and_exit(banner, nil,  $\text{'Tomas\_Rokicki'}$ , 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  $\neq$  argc)  $\wedge$  (optind + 2  $\neq$  argc) then
    begin write_ln(stderr, my_name,  $\text{'\_Need\_one\_or\_two\_file\_arguments.'}$ ); 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*⟩  $\equiv$ 
  current_option  $\leftarrow$  0; long_options[current_option].name  $\leftarrow$   $\text{'help'}$ ;
  long_options[current_option].has_arg  $\leftarrow$  0; long_options[current_option].flag  $\leftarrow$  0;
  long_options[current_option].val  $\leftarrow$  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*⟩  $\equiv$ 
  long_options[current_option].name  $\leftarrow$   $\text{'version'}$ ; long_options[current_option].has_arg  $\leftarrow$  0;
  long_options[current_option].flag  $\leftarrow$  0; long_options[current_option].val  $\leftarrow$  0; incr(current_option);

```

77* Print progress information?

```

⟨Define the option table 75*⟩  $\equiv$ 
  long_options[current_option].name  $\leftarrow$   $\text{'verbose'}$ ; long_options[current_option].has_arg  $\leftarrow$  0;
  long_options[current_option].flag  $\leftarrow$  address_of(verbose); long_options[current_option].val  $\leftarrow$  1;
  incr(current_option);

```

78* ⟨Globals in the outer block 11⟩ \equiv

```

verbose: c_int_type;

```

79* \langle Initialize the option variables 79* $\rangle \equiv$
verbose \leftarrow *false*;

This code is used in section 74*.

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

\langle Define the option table 75* $\rangle + \equiv$
long_options[*current_option*].*name* \leftarrow 0; *long_options*[*current_option*].*has_arg* \leftarrow 0;
long_options[*current_option*].*flag* \leftarrow 0; *long_options*[*current_option*].*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](#).
 boolean: [62](#), [67](#).
 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*](#)
 cur_n: [65*](#), [66](#), [67](#).
 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](#).
 end_of_packet: [47](#), [48](#), [52](#), [53](#), [54](#).
 endcases: [3](#).
 eoc: [14](#), [16](#), [17](#), [18](#), [47](#).
 eof: [40*](#), [43*](#), [73*](#)
 false: [64](#), [79*](#)
 first_on: [65*](#), [66](#), [67](#).
 first_text_char: [10*](#), [13](#).
 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_nyb: [30*](#), [62](#).
 get_three_bytes: [43*](#)
 get_two_bytes: [43*](#), [45*](#)
 get_16: [45*](#), [53](#), [54](#).
 get_32: [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](#).
 gf_16: [46*](#), [66](#).
 gf_24: [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](#)*
integer: [4](#)*, [30](#)*, [41](#)*, [43](#)*, [46](#)*, [48](#), [50](#), [51](#)*, [55](#), [57](#), [62](#),
[63](#)*, [65](#)*, [67](#), [69](#), [70](#), [74](#)*
j: [48](#).
Japanese characters: [19](#).
Knuth, D. E.: [29](#).
kpse_find_pk: [40](#)*
kpse_init_prog: [4](#)*
kpse_open_file: [40](#)*
kpse_pk_format: [40](#)*
kpse_set_program_name: [4](#)*
last_eoc: [47](#), [49](#)*, [55](#), [68](#).
last_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](#)*
n_options: [74](#)*
name: [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](#).
open_gf_file: [40](#)*, [44](#).
open_pk_file: [40](#)*, [44](#).
optind: [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](#)*
pk_byte: [38](#), [43](#)*, [49](#)*, [53](#), [54](#), [62](#), [70](#), [73](#)*
pk_file: [39](#), [40](#)*, [43](#)*, [73](#)*
pk_id: [24](#), [49](#)*
pk_loc: [41](#)*, [43](#)*, [47](#), [52](#), [53](#), [54](#), [73](#)*
pk_name: [40](#)*, [41](#)*
pk_no_op: [23](#), [24](#).
pk_packed_num: [30](#)*, [62](#), [64](#).
pk_post: [23](#), [24](#), [70](#), [73](#)*
pk_pre: [23](#), [24](#), [49](#)*
pk_xxx1: [23](#), [24](#).
pk_yyy: [23](#), [24](#).
PKtoGF: [4](#)*
PKTOGF_HELP: [74](#)*
pl: [32](#).
post: [14](#), [16](#), [17](#), [18](#), [20](#), [68](#).
post_post: [16](#), [17](#), [18](#), [20](#), [68](#).
pre: [14](#), [16](#), [17](#), [49](#)*
preamble_comment: [2](#)*
print: [4](#)*, [49](#)*
print_ln: [4](#)*, [8](#)*, [49](#)*, [60](#), [65](#)*, [73](#)*
print_version_and_exit: [74](#)*
proofing: [19](#).
put_byte: [42](#)*
rcp: [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](#).

text_char: [10](#)*, [11](#).
text_file: [10](#)*.
tfm: [32](#), [33](#), [36](#).
tfm_width: [48](#), [52](#), [53](#), [54](#), [60](#).
this_char_ptr: [57](#), [59](#), [70](#).
true: [8](#)*, [64](#).
turn_on: [47](#), [64](#), [65](#)*, [66](#), [67](#).
uexit: [8](#)*.
undefined_commands: [17](#).
usage: [74](#)*.
usage_help: [74](#)*.
val: [75](#)*, [76](#)*, [77](#)*, [80](#)*.
ver_esc: [52](#), [53](#), [54](#), [55](#), [60](#).
verbose: [4](#)*, [8](#)*, [77](#)*, [78](#)*, [79](#)*.
voff: [32](#), [34](#).
vppp: [18](#), [23](#), [49](#)*, [50](#), [68](#).
white: [16](#).
width: [31](#).
word_width: [52](#), [53](#), [54](#), [55](#).
write: [4](#)*.
write_ln: [4](#)*, [74](#)*.
x_off: [48](#), [52](#), [53](#), [54](#), [56](#).
x_to_go: [65](#)*, [67](#).
xchr: [11](#), [12](#), [13](#), [49](#)*.
xmalloc_array: [51](#)*.
xord: [11](#), [13](#), [49](#)*.
xrealloc_array: [65](#)*.
xxx1: [16](#), [17](#).
xxx2: [16](#).
xxx3: [16](#).
xxx4: [16](#).
y_off: [48](#), [52](#), [53](#), [54](#), [56](#).
y_to_go: [65](#)*, [66](#), [67](#).
yyy: [16](#), [17](#), [19](#), [23](#).

〈Calculate and check *min_m*, *max_m*, *min_n*, and *max_n* 56〉 Used in section 47.
 〈Constants in the outer block 6*〉 Used in section 4*.
 〈Define *parse_arguments* 74*〉 Used in section 4*.
 〈Define the option table 75*, 76*, 77*, 80*〉 Used in section 74*.
 〈Get next count value into *count* 64〉 Used in section 65*.
 〈Globals in the outer block 11, 39, 41*, 48, 50, 55, 57, 63*, 67, 69, 78*〉 Used in section 4*.
 〈Initialize the option variables 79*〉 Used in section 74*.
 〈Open files 44〉 Used in section 73*.
 〈Output row 66〉 Used in section 65*.
 〈Packed number procedure 30*〉 Used in section 62.
 〈Read and translate raster description 65*〉 Used in section 47.
 〈Read extended short character preamble 53〉 Used in section 47.
 〈Read long character preamble 52〉 Used in section 47.
 〈Read preamble 49*〉 Used in section 73*.
 〈Read short character preamble 54〉 Used in section 47.
 〈Save character locator 60〉 Used in section 47.
 〈Set initial values 12, 13, 51*, 58〉 Used in section 4*.
 〈Types in the outer block 9, 10*, 38〉 Used in section 4*.
 〈Unpack and write character 47〉 Used in section 73*.
 〈Write character locators 61〉 Used in section 68.
 〈Write character preamble 59〉 Used in section 47.
 〈Write GF postamble 68〉 Used in section 73*.