2* The *banner* string defined here should be changed whenever *PKtype* gets modified.

\[
\text{define } my\_name \equiv \textquotesingle pktype\textquotesingle \\
\text{define } banner \equiv \textquotesingle This\_is\_PKtype,\_Version\_2.3\textquotesingle \{ \text{printed when the program starts} \}
\]

4* Both the input and output come from binary files. On line interaction is handled through Pascal’s standard *input* and *output* files. Two macros are used to write to the type file, so this output can easily be redirected.

\[
\text{define } print\_ln(\#) \equiv \text{write}\_ln(output,\#) \\
\text{define } print(\#) \equiv \text{write}(output,\#) \\
\text{define } typ\_file \equiv \text{stdout} \\
\text{define } t\_print\_ln(\#) \equiv \text{write}\_ln(typ\_file,\#) \\
\text{define } t\_print(\#) \equiv \text{write}(typ\_file,\#)
\]

program *PKtype*(input, output);

\[
\text{type} \{ \text{Types in the outer block} \} \\
\text{var} \{ \text{Globals in the outer block} \}
\]

\[
\text{procedure } \text{initialize}; \{ \text{this procedure gets things started properly} \} \\
\text{var } i : \text{integer}; \{ \text{loop index for initializations} \} \\
\text{begin } \text{kpse}\_\text{set}\_\text{program}\_\text{name}(argv[0], my\_name); \text{kpse}\_\text{init}\_\text{prog}(\textquotesingle PKTYPE\textquotesingle, 0, nil, nil); \\
\text{parse}\_\text{arguments}; \text{print}(banner); \text{print}\_\text{ln}(version\_\text{string}); \\
\text{end}; \{ \text{Set initial values} \}
\]

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 widest character that can be translated.

8* We use a call to the external C exit to avoid a non-local *goto*.

\[
\text{define } abort(\#) \equiv \\
\text{ \hspace{1cm} begin } \text{print}\_\text{ln}(\#); \text{uexit}(1) \text{ end}
\]
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 PKtype. So we shall assume that the Pascal system being used for PKtype 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.

\[
\begin{align*}
\text{define } & \text{ char } \equiv 0 \ldots 255 & \{ \text{the data type of characters in text files} \} \\
\text{define } & \text{ text_char } \equiv \text{ char} & \{ \text{ordinal number of the smallest element of text_char} \} \\
\text{define } & \text{ first_text_char } = 0 & \{ \text{ordinal number of the largest element of text_char} \} \\
\text{define } & \text{ last_text_char } = 127 & \{ \text{ordinal number of the largest element of text_char} \}
\end{align*}
\]

\[\text{Types in the outer block } 9 + \equiv\]

\[
\text{text_file } = \text{ packed file of } \text{ text_char};
\]
31* ⟨Globals in the outer block 11⟩ $\equiv$

$pk_{file}$: $byte_{file}$;  { where the input comes from }

32* In C, do path searching.

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. }
    $pk_{file}$ ← $kpse_open_{file}$(cmdline(1), $kpse_{pk}_{format}$); $cur_{loc}$ ← 0;
  end;

33* We need a place to store the names of the input and output file, as well as a byte counter for the output file.

 ⟨Globals in the outer block 11⟩ $\equiv$

$pk_{name}$: c_string;  { name of input and output files }
$cur_{loc}$: integer;  { how many bytes have we read? }
34* 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. We comment out the ones we don’t need.

\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 \times 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 \times 256 + b
else signed\_pair \leftarrow (a - 256) \times 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 \times 256 + b) \times 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 \times 256 + b) \times 256 + c
else signed\_trio \leftarrow ((a - 256) \times 256 + b) \times 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 \times 256 + b) \times 256 + c) \times 256 + d
else signed\_quad \leftarrow (((a - 256) \times 256 + b) \times 256 + c) \times 256 + d;
end;
\end{verbatim}
This module was needed when output was directed to *typ_file*. It is not needed when output goes to *stdout*.

As we are reading the packed file, we often need to fetch 16 and 32 bit quantities. Here we have two procedures to do this.

```
define get_16 ≡ get_two_bytes
define get_32 ≡ signed_quad
```
§40  PK type changes for C

7

52* If any specials are found, we write them out here.

define four_cases(#) ≡ #, # + 1, # + 2, # + 3

procedure skip_specials;
  var i, j: integer;
  begin repeat
    flag_byte ← pk_byte;
    if flag_byte ≥ 240 then
      case flag_byte of
        four_cases(pk_xxx1): begin
          t_print((pk_loc - 1): 1, ´;Special:´´); i ← 0;
          for j ← pk_xxx1 to flag_byte do i ← 256 * i + pk_byte;
          for j ← 1 to i do t_print(xchr[pk_byte]);
          t_print_ln(´´); end;
        pk_yyy: begin
          t_print((pk_loc - 1): 1); t_print_ln(´;Num_special:´, get_32: 1);
          end;
        pk_post: t_print_ln((pk_loc - 1): 1, ´;Postamble´);
        pk_no_op: t_print_ln((pk_loc - 1): 1, ´;No_op´);
        pk_pre, pk_undefined: abort(´Unexpected, flag_byte : 1, ´!´);
      endcases;
    until (flag_byte < 240) ∨ (flag_byte = pk_post);
  end;
53* Terminal communication. There isn’t any.

54* So there is no procedure dialog.
55* The main program. Now that we have all the pieces written, let us put them together.

begin initialize; open pk_file; ⟨Read preamble 38⟩;
skip_specials;
while flag_byte ≠ pk_post do
  begin ⟨Unpack and write character 40⟩;
    skip_specials;
  end;
  j ← 0;
while ¬eof (pk_file) do
  begin i ← pk_byte;
    if i ≠ pk_no_op then abort(‘Bad byte at end of file; ′, i : 1);
    t.print ln((pk_loc − 1) : 1, ´: No op’); incr(j);
  end;
  t.print ln(pk_loc : 1, ´bytes read from packed file.’);
end.
**56** System-dependent changes. Parse a Unix-style command line.

```plaintext
define argument_is(#) ≡ (strcmp(long_options[option_index].name,#) = 0)

(Define parse_arguments 56* ) ≡
```

```plaintext
procedure parse_arguments;

const n_options = 2; { Pascal won’t count array lengths for us. }

var long_options: array [0..n_options] of getopt_struct;

begin (Define the option table 57* );

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

else if getopt_return_val = ’?’ then

begin usage(my_name);

end

else if argument_is(’help’) then

begin usage_help(PKTYPE_HELP, nil);

end

else if argument_is(’version’) then

begin print_version_and_exit(banner, ’Tomas_Rokicki’ , nil);

end: { Else it was just 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. }

if (optind + 1 ≠ argc) then

begin write Ln(stderr, my_name, ’⋯I need exactly one file argument. ’); usage(my_name);

end;

end;
```

This code is used in section 4*.

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

```plaintext
(Define the option table 57* ) ≡

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 58* and 59*.

This code is used in section 56*.

**58** Another of the standard options.

```plaintext
(Define the option table 57* ) +≡

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);
```

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

```plaintext
(Define the option table 57* ) +≡

long_options[current_option].name ← 0; long_options[current_option].has_arg ← 0;

long_options[current_option].flag ← 0; long_options[current_option].val ← 0;
```
60* 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, 31, 32, 33, 34, 35, 36, 52, 53, 54, 55, 56, 57, 58, 59, 60.

-help:  57*
-version:  58*
a:  31*
abort:  8*, 23, 38, 40, 50, 52*, 55*
address_of:  56*
argv:  56*
argument_is:  56*
argv:  4*, 56*
ASCII_code:  9, 11.
b:  34*
Bad byte at end of file:  55*
Bad packet length:  40.
banner:  2*, 4*, 56*
bit_weight:  45, 47, 48.
boolean:  41, 45, 46, 51.
byte_file:  30, 31*
c:  34*
c_int_type:  56*
c_string:  33*
car:  40, 41, 42, 43, 44.
cc:  25.
char:  10*
checksum:  38, 39.
chr:  10*, 11, 13.
cmdline:  32*
count:  50, 51.
cs:  16.
cur_loc:  32*, 33*, 34*
current_option:  56*, 57*, 58*, 59*
d:  34*
decr:  7, 23.
design_size:  38, 39.
dialog:  54*
dm:  25.
do_nothing:  7, 56*
ds:  16.
dx:  25, 40, 41, 42, 43, 44.
dxs:  41.
dy:  25, 40, 41, 42, 43, 44.
dys:  41.
dys:  41.
eight_bits:  30, 34*, 45, 47.
else:  3.
end:  3.
end_of_packet:  40, 41, 42, 43, 44.
endcases:  3.
eof:  34*, 55*
false:  50.

flag:  25, 57*, 58*, 59*
flag_byte:  40, 41, 43, 44, 52*, 55*
four_cases:  52*
get_bit:  45, 49.
get_byte:  34*
get_nyb:  23, 45.
get_three_bytes:  34*
get_two_bytes:  34*, 36*
get_16:  36*, 43, 44.
get_32:  36*, 38, 42, 52*
g getopt:  56*
g getopt_long_only:  56*
g getopt_return_val:  56*
g getopt_struct:  56*
h_bit:  50, 51.
has_arg:  57*, 58*, 59*
height:  24, 40, 41, 42, 43, 44, 49, 50.
hoff:  25, 27.
hppp:  16, 38, 39.
i:  4*, 23, 41, 46, 52*
ingr:  7, 23, 34*, 46, 55*, 57*, 58*
initialize:  4*, 55*
input:  4*
input_byte:  45, 47.
integer:  4*, 23, 33*, 34*, 37, 39, 41, 45, 46, 51, 52*, 56*
j:  23, 41, 52*
Knuth, Donald Ervin:  22.
kpse_find_pk:  32*
kpse_init_prog:  4*
k pse_open_file:  32*
k pse_pk_format:  32*
k pse_set_program_name:  4*
len:  46.
long_options:  56*, 57*, 58*, 59*
magnification:  38, 39.
More bits than required:  50.
my_name:  2*, 4*, 56*
n_options:  56*
name:  56*, 57*, 58*, 59*
nybble:  47.
open_pk_file:  32*, 55*
op tind:  56*
op tion_index:  56*
ord:  11.
othercases:  3.
others:  3.
output:  4*
packet_length:  40, 41, 42, 43, 44.
parse_arguments:  4*, 56*
pk_byte:  30, 34*, 38, 43, 44, 45, 52*, 55*
pk_file:  31*, 32*, 34*, 55*
pk_id:  17, 38.
pk_loc:  34*, 40, 42, 43, 44, 52*, 55*
pk_name:  33*,
pk_no_op:  16, 17, 52*, 55*,
pk_packed_num:  23, 50.
pk_post:  16, 17, 52*, 55*
pk_pre:  16, 17, 38, 52*
pk_undefined:  17, 52*
pk_xxx1:  16, 17, 52*
pk_yyy:  16, 17, 52*
PKtype:  4*
PKTYPE_HELP:  56*
pl:  25.
pre command missing:  38.
print:  4*
print_ln:  4*, 8*, 38.
print_version_and_exit:  56*
read:  34*
repeat_count:  23, 46, 50, 51.
round:  38.
rows_left:  50, 51.
scaled:  16.
Second repeat count...:  23.
send_out:  23, 46, 50.
signed_byte:  34*
signed_pair:  34*
signed_quad:  34*, 36*
signed_trio:  34*
skip_specials:  52*, 55*
status:  41.
stderr:  56*
stdout:  4*, 35*,
strcmp:  56*

system dependencies:  6*, 10*, 30, 31*, 34*
t_print:  4*, 38, 40, 46, 49, 50, 52*
t_print_ln:  4*, 38, 40, 46, 49, 50, 52*, 55*
temp:  45.
term_pos:  37, 46, 50.
text_char:  10*, 11.
text_file:  10*,
tfm:  25, 26, 29.
tfm_width:  40, 41, 42, 43, 44.
tfms:  41.
true:  23.
turn_on:  40, 46, 50, 51.
vecexit:  8*

Unexpected bbb:  52*

usage:  56*
(Create normally packed raster 50) Used in section 48.
(Define the option table 57*, 58*, 59*) Used in section 56*.
(Define parse_Arguments 56*) Used in section 4*.
(Get raster by bits 49) Used in section 48.
(Globals in the outer block 11, 31*, 33*, 37, 39, 41, 47, 51) Used in section 4*.
(Packed number procedure 23) Used in section 46.
(Read and translate raster description 48) Used in section 40.
(Read extended short character preamble 43) Used in section 40.
(Read long character preamble 42) Used in section 40.
(Read preamble 38) Used in section 55*.
(Read short character preamble 44) Used in section 40.
(Set initial values 12, 13) Used in section 4*.
(Types in the outer block 9, 10*, 30) Used in section 4*.
(Unpack and write character 40) Used in section 55*.