

The PKtype processor

(Version 2.3, 23 April 2020)

	Section	Page
Introduction	1	2
The character set	9	3
Packed file format	14	4
Input and output	30	4
Character unpacking	40	7
Terminal communication	53	8
The main program	55	9
System-dependent changes	56	10
Index	60	11

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](#), [31](#), [32](#), [33](#), [34](#), [35](#), [36](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#).

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 **PKtype** gets modified.

```
define my_name  $\equiv$  `pktype`
define banner  $\equiv$  `This_is_PKtype,_Version_2.3` { 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.

```
define print_ln(#)  $\equiv$  write_ln(output, #)
define print(#)  $\equiv$  write(output, #)
define typ_file  $\equiv$  stdout
define t_print_ln(#)  $\equiv$  write_ln(typ_file, #)
define t_print(#)  $\equiv$  write(typ_file, #)

program PKtype(input, output);
type  $\langle$  Types in the outer block 9  $\rangle$ 
var  $\langle$  Globals in the outer block 11  $\rangle$ 
   $\langle$  Define parse_arguments 56*  $\rangle$ 
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(`PKTYPE`, 0, nil, nil);
  parse_arguments; print(banner); print_ln(version_string);
   $\langle$  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 widest character that can be translated.

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

```
define abort(#)  $\equiv$ 
  begin print_ln(#); 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 **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.

```
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 ⟩ \equiv

```
text_file = packed file of text_char;
```

31* \langle Globals in the outer block 11 $\rangle + \equiv$

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 \leftarrow *kpse_open_file*(*cmdline*(1), *kpse_pk_format*); *cur_loc* \leftarrow 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.

\langle Globals in the outer block 11 $\rangle + \equiv$

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 *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.

```

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;

```

35* This module was needed when output was directed to *typ_file*. It is not needed when output goes to *stdout*.

36* 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* \equiv *get_two_bytes*

define *get_32* \equiv *signed_quad*

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

```

define four_cases(#)  $\equiv$  #, # + 1, # + 2, # + 3
procedure skip_specials;
  var i, j: integer;
  begin repeat flag_byte  $\leftarrow$  pk_byte;
    if flag_byte  $\geq$  240 then
      case flag_byte of
        four_cases(pk_xxx1): begin t_print((pk_loc - 1) : 1, ^: Num_Special: ^); i  $\leftarrow$  0;
          for j  $\leftarrow$  pk_xxx1 to flag_byte do i  $\leftarrow$  256 * i + pk_byte;
          for j  $\leftarrow$  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)  $\vee$  (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, `:Noop`); 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.

```

define argument_is(#)  $\equiv$  (strcmp(long_options[option_index].name, #) = 0)
⟨Define parse_arguments 56*⟩  $\equiv$ 
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;
    getopt_return_val: integer; option_index: c_int_type; current_option: 0 .. n_options;
  begin ⟨Define the option table 57*⟩;
  repeat getopt_return_val  $\leftarrow$  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, nil, `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  $\neq$  argc) then
    begin write_ln(stderr, my_name, `: 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.

```

⟨Define the option table 57*⟩  $\equiv$ 
  current_option  $\leftarrow$  0; long_options[current_option].name  $\leftarrow$  `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 58* and 59*.

This code is used in section 56*.

58* Another of the standard options.

```

⟨Define the option table 57*⟩  $\equiv$ 
  long_options[current_option].name  $\leftarrow$  `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);

```

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

```

⟨Define the option table 57*⟩  $\equiv$ 
  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;

```

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](#): [34*](#)
[abort](#): [8*](#), [23](#), [38](#), [40](#), [50](#), [52*](#), [55*](#)
[address_of](#): [56*](#)
[argc](#): [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](#).
[dyn_f](#): [21](#), [22](#), [23](#), [24](#), [25](#), [28](#), [29](#), [40](#), [41](#), [48](#), [49](#).
[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](#).
[first_text_char](#): [10*](#), [13](#).
[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*](#)
[getopt](#): [56*](#)
[getopt_long_only](#): [56*](#)
[getopt_return_val](#): [56*](#)
[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*](#)
[incr](#): [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*](#)
[kpse_open_file](#): [32*](#)
[kpse_pk_format](#): [32*](#)
[kpse_set_program_name](#): [4*](#)
[last_text_char](#): [10*](#), [13](#).
[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*](#)
[optind](#): [56*](#)
[option_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.
typ_file: 4*, 35*, 40.
uexit: 8*
 Unexpected bbb: 52*
usage: 56*
usage_help: 56*
val: 57*, 58*, 59*
value: 46.
version_string: 4*
voff: 25, 27.
vppp: 16, 38, 39.
width: 24, 40, 41, 42, 43, 44, 49, 50.
write: 4*
write_ln: 4*, 56*
 Wrong version of PK file: 38.
x_off: 40, 41, 42, 43, 44.
xchr: 11, 12, 13, 38, 52*
xord: 11, 13.
y_off: 40, 41, 42, 43, 44.
yyy: 16.

⟨ Create normally packed raster 50 ⟩ Used in section 48.
⟨ Define *parse_arguments* 56* ⟩ Used in section 4*.
⟨ Define the option table 57*, 58*, 59* ⟩ Used in section 56*.
⟨ 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*.