IV. Pascal Internal Subroutines

NAME

Conventions - the Pascal runtime

SYNOPSIS

```
#include <pascal.h>
```

FUNCTION

The Pascal runtime support library is written entirely in C, not necessarily callable directly from a Pascal program. It is not necessary to know about any of these functions to make use of the Pascal system; the material in this section is aimed at those who must maintain the Pascal runtime, or who must displace one or more of its functions. Consequently, the manual pages are written for the C programmer, not in terms of Pascal; and they are designed to provide a complete specification of the facilities, outside the portable C library, that may be called upon by the Pascal translator. The C Programmers' Manual describes, in equal detail, any additional functions that may be used.

A number of routines must conspire to perform input/output by the rules of Pascal; these share common conventions by including the header file pascal.h from the system header file collection. Each open Pascal file is controlled by an element of the array p_files, whose structure is given by the type definition PFILE in the header file. Other definitions provide symbolic values for the elements of PFILE.

The PFILE structure consists of the fields:

p_buf -- the address of the corresponding file variable, or NULL if the element is idle.

p_size - the size of the file variable in bytes, or zero if a text file. A text file variable is known to be one byte.

p_mode - the state of the open file, which may have any of the values (P_EOF, P_EOLN, P_INVAL, P_VALID, P_WRITE, P_WROTE).

p_fd - the file descriptor associated with the file. If the P_TFD bit is set in this field, the file is temporary and should be removed when closed.

The values of p_mode are:

P_EOF - file is open for reading and is at end of file.

P_EOLN - file is open for reading as text and the buffer contains a space corresponding to an end of line.

P_INVAL - file is open for reading and the buffer is not valid, i.e. it does not correspond to the next record to be read.

P_VALID - file is open for reading and the buffer is valid.

P_WRITE - file is open for writing and no incomplete text line is pending.

P_WROTE - file is open for writing as text and the last character written was not a newline.

The size of the array p_files, and hence the maximum number of files that may be simultaneously open, is given by NFILES. This value is currently 16, which is more files than most operating systems will support.

NAME

Mapping - Pascal to C

FUNCTION

This implementation of Pascal is effectuated by a) translating Pascal statements wherever possible to equivalent C statements, and b) emitting calls to a supporting runtime library for those operations which are difficult or impossible to express in C. (The runtime library is detailed in the remainder of this section.) To intermix source files in the two languages, or merely to understand how the translator does its job, one must have some notion of how Pascal is represented in the C source file output by the translator. Herewith a summary of the most important features:

DATA REPRESENTATION

Enumerations - are represented as integer subranges starting at zero. Thus, the builtin type boolean, which is the enumeration (false, true), is represented as the subrange 0..1; and the builtin type char becomes the subrange 0..255. Subranges map into C types by choosing the earliest acceptable entry in the table:

Subrange	C Type
[-128, 128)	char
[0, 256)	unsigned char
[-32768, 32768)	short
[0, 65536)	unsigned short
anything else	long

Programmer defined enumerations almost invariably are represented as char variables.

Sets - of up to eight elements are represented as char variables; 9-16 element sets are short integers; and 17-32 element sets are longs. Any larger sets become arrays of char that hold eight set elements in each array element. In all cases, the least significant bit in a char or larger int is associated with the lowest numbered set element; a bit is set if the associated element is present in the set.

Reals - are doubles, unless the -f flag is specified to the translator. The -f flag causes reals to be translated to float variables.

Pointers - are pointers.

Arrays - are arrays. The attribute packed has no effect.

Records - are structs, with the fields in the same order as in the record declaration. A variant part becomes a union within the struct. The attribute packed has no effect.

Files - are variables of the underlying type. The variable serves as the file buffer, and its address provides the unique identifier by which the runtime functions associate open file status with a given file variable. The Pascal type text becomes a char variable, with the added understanding that I/O may be mapped as needed to satisfy local custom for representing readable text (e.g., carriage returns may be deleted on input, inserted on output). The Pascal type file of char also becomes a char variable, but I/O is "binary", i.e. eight-bit transparent except for the possible addition of one or more trailing NULs to a binary file. file of anything else also calls for binary I/O, with multi-byte data represented in native byte order. Since "lazy input" is used to provide better behavior on interactive I/O, the buffer contents are not necessarily valid even when they could be; various runtime routines are used to make buffers current before reading.

Procedures - are functions that return no result.

Functions - are functions.

Naturally, these mappings are applied recursively. The best way to learn the implications of all these rules is to stuff a nontrivial Pascal program through the translator and to study the emissions.

EXTERNAL DECLARATIONS

To match up declarations across multiple files, this implementation of Pascal provides the fundamental rules:

imports - any identifier mentioned in the parenthesized list in the program statement is assumed to be a var declaration defined in some other source file and imported for reference in this file. An exception is any such identifier, imported to a Pascal main program (one with a program identifier provided in the program statement), and explicitly declared to be of type file. In this case, the declaration is taken as the defining instance of the file variable, so no other definition is needed or permitted. If there is no explicit declaration (in the outermost set of var declarations) for an identifier in the parenthesized list of a program statement, the identifier is implicitly imported as a var of type text, defined in some other file. Note that the runtime library provides defining instances of the text variables input and output.

externals - any function or procedure declaration containing the directive external in lieu of a defining body is assumed to be a function defined in some other file and imported for reference in this file.

exports - any file containing no program identifier in the program statement is assumed to be a "library file". It must contain no main program; instead, all of its outermost level of var declarations, functions, and procedures not explicitly imported are defining instances that are exported for use with import lists (or external declarations) in other source files. A given identifier may be exported by only one library file.

Thus, Pascal variables, functions, and procedures to be referenced in multiple files must be defined in library files. Variables are imported via the program list, and functions or procedures via declarations with the directive external.

Note that there is no provision for expressing a static initializer in a Pascal variable declaration. A defining instance is always initialized to zero.

FUNCTION CALLS

Since Pascal functions, like those in C, may return only scalar values, the correspondence between return values is straightforward. Pascal has a number of ways of passing arguments, however, that require some special handling in the C world. A parameter with the attribute var, for instance, is converted to a pointer to the actual parameter. Even in the absence of the attribute var, however, a pointer must often be passed; to wit:

arrays or records - are passed by reference (i.e. the argument is a pointer to the actual parameter), then copied into an auto temporary on function entry.

sets - of more than 32 elements are treated the same as arrays.

files - are passed by reference, but buffer contents are not copied in on function entry.

functions or procedures - are passed by reference, with a second pointer argument to indicate the current dynamic instance of the lexically enclosing block. If there is no such outer block, this second argument is still present, but NULL. When the function or procedure is called, the outer block pointer is passed as an extra argument (after all the declared arguments), so the function can merely ignore it if it is irrelevant.

Since any Pascal function callable from C will have no lexically enclosing block, no attempt is made here to explain the dynamic chaining mechanism.

In a nutshell, scalar parameters are passed by value whenever possible, while var and nonscalar parameters are of necessity passed by reference.

NAME

_pargs - array of program arguments

SYNOPSIS

```
struct {
    TEXT *var, *name;
    } _pargs[];
```

FUNCTION

_pargs is an array provided by each Pascal main file to record the addresses and names of all arguments on the program line. _pargs[i].var is the address of the variable corresponding to argument i on the program line, counting from one; a NULL value signals the end of the array. _pargs[i].name points at the NUL terminated string which is the identifier corresponding to argument i; it is in uppercase.

The program argument array is used by p_pnam to concoct a filename if there is no command line argument corresponding to that parameter.

SEE ALSO

p_pnam

BUGS

The Pascal translator should provide the names in lowercase.

NAME

chr - integer to char

SYNOPSIS

```
TEXT chr(i)
    BYTES i;
```

FUNCTION

chr returns its integer argument as a character, a trivial operation. It is provided as a library function in case chr is used as a function parameter on a function or procedure call.

RETURNS

chr returns its integer argument as the value of the function.

NAME

eof - test for end of file

SYNOPSIS

```
BOOL eof(pfile)
    TEXT *pfile;
```

FUNCTION

eof tests whether end of file has been encountered on the file associated with the variable at pfile. It performs lazy input, if necessary, to make the test.

RETURNS

eof returns true if the file is opened for writing, or if the file is opened for reading and is positioned beyond its last information. It returns false if the file is opened for reading and there is more information available. Otherwise it aborts.

SEE ALSO

eoln

NAME

eoln - test for end of line

SYNOPSIS

```
BOOL eoln(pfile)
    TEXT *pfile;
```

FUNCTION

eoln tests whether end of line has been encountered on the text file associated with the variable at pfile. It performs lazy input, if necessary, to make the test.

RETURNS

eoln returns true if the text file is opened for reading and the character in the file buffer is the space associated with an end of line condition. It returns false if the text file is opened for reading and is not positioned at end of line. Otherwise it aborts.

SEE ALSO

eof

NAME

iabs - integer absolute value

SYNOPSIS

```
BYTES iabs(i)
    ARGINT i;
```

FUNCTION

iabs computes the absolute value of i. No check is made for overflow.

RETURNS

iabs returns the absolute value of its signed integer argument.

NAME

input - standard input file

SYNOPSIS

```
TEXT input;
```

FUNCTION

input is the file variable used by default on eof, eoln, read, and readln calls in Pascal. There is an initial entry in p_files that makes input open for reading from the file descriptor STDIN, with an invalid buffer.

input is packaged with p_ckfd.

SEE ALSO

output, p_ckfd, p_files

NAME

isqr - integer square

SYNOPSIS

```
BYTES isqr(i)
    BYTES i;
```

FUNCTION

isqr computes the square of i. No check is made for overflow.

RETURNS

isqr returns the square of its integer argument.

NAME

main - enter Pascal program

SYNOPSIS

```
BOOL main(ac, av)
    BYTES ac;
    TEXT **av;
```

FUNCTION

main is the function that gets control from the standard C environment, and calls the Pascal main program in turn. The command line used to invoke the program is represented as an array of ac pointers to NUL terminated strings, the pointers stored at av[0], av[1], etc. av[0] is the name by which the program was invoked, if known; otherwise it is the program name from the Pascal program header for the main file.

main sets the external variable argc to ac, the external variable argv to av, enters the Pascal main, then closes all Pascal files upon return.

RETURNS

main always returns success.

NAME

odd - test for odd integer

SYNOPSIS

```
BOOL odd(i)
    BYTES i;
```

FUNCTION

odd tests whether its integer argument is odd, i.e. whether its least significant bit is set.

RETURNS

odd simply returns (i & 1).

NAME

output - standard output file

SYNOPSIS

```
TEXT output;
```

FUNCTION

output is the file variable used by default on page, write, and writeln calls in Pascal. There is an initial entry in p_files that makes output open for writing to the file descriptor STDOUT.

output is packaged with p_ckfd.

SEE ALSO

input, p_ckfd, p_files

NAME

p_abort - print message and abort

SYNOPSIS

```
VOID p_abort(name, mesg)
    TEXT *name, *mesg;
```

FUNCTION

p_abort writes an error message to STDERR, then takes an error exit. The message is the single line

```
<_pname>: <name> -- <mesg>
```

where <_pname> is the name by which the program was invoked, and <name> and <mesg> are the NUL terminated argument strings, name and mesg.

RETURNS

p_abort never returns to its caller.

NAME

p_bget - get binary records

SYNOPSIS

```
VOID p_bget(pfile, fmt, arg1, ...)
    TEXT *pfile, *fmt;
    TEXT *arg1, ...;
```

FUNCTION

p_bget reads binary records, from the file associated with the variable at pfile, into the record areas beginning at arg1, ... A record is read for each character in the NUL terminated string at fmt. (The format characters use the same code as for p_fget, but since all records must be identical, the actual codes are ignored.)

RETURNS

Nothing. If the file is not readable, it aborts.

SEE ALSO

p_fget, p_tget

NAME

p_bput - put binary records

SYNOPSIS

```
VOID p_bput(pfile, fmt, arg1, ...)
    TEXT *pfile, *fmt;
    ...;
```

FUNCTION

p_bput writes binary records, to the file associated with the variable at pfile, from the arguments arg1, ... A record is written for each character in the NUL terminated string at fmt, by the following rules:

a, b, c - cause the next argument to be taken as an integer and written as a character integer.

s - causes the next argument to be taken as an integer and written as a short integer.

l - causes the next argument to be taken as a long integer and written as a long integer.

' - causes the next argument to be taken as a pointer to an array of characters, whose length is specified by the integer argument after that. The array is written.

f - causes the next argument to be taken as a double and written as a float.

d - causes the next argument to be taken as a double and written as a double.

p, v - causes the next argument to be taken as a pointer to an array of characters, whose length is specified by the size of the file buffer.

If any other character is present, or if the size of any argument does not match the size of the file buffer, then p_bput performs a panic abort, since the Pascal translator should never generate such a call.

RETURNS

Nothing.

SEE ALSO

p_fput, p_tput

NAME

p_ckfd - check Pascal file variable

SYNOPSIS

```
PFILE *p_ckfd(pfile, name)
    TEXT *pfile, *name;
```

FUNCTION

p_ckfd scans the array p_files for an entry whose p_buf entry matches the pointer to file variable pfile. If it fails to find such an entry, and if name is not NULL, then it aborts, using name as the name of the aborting program.

The array p_files and the standard files input and output are packaged with p_ckfd.

RETURNS

p_ckfd returns the address of the entry in p_files, if found, otherwise NULL.

NAME

p_close - close Pascal files

SYNOPSIS

```
VOID p_close(plo, range)
    TEXT *plo;
    BYTES range;
```

FUNCTION

p_close closes all Pascal files whose associated file variables begin in the interval [plo, plo+range). If a file to be closed is marked as temporary, by P_TFD in the p_fd field, then it is removed after being closed.

RETURNS

Nothing.

NAME

p_cmp - compare two buffers

SYNOPSIS

```
COUNT p_cmp(n, l, r)
    BYTES n;
    UTINY *l, *r;
```

FUNCTION

p_cmp performs a byte by byte comparison of the n character buffers starting at l and at r.

RETURNS

p_cmp returns zero if the two buffers are equal. Otherwise it returns the difference between the leftmost two bytes that differ; a negative value implies that the left buffer is lower in value than the right buffer.

NAME

p_copy - copy a buffer

SYNOPSIS

```
VOID p_copy(n, l, r)
    BYTES n;
    TEXT *l, *r;
```

FUNCTION

p_copy copies the n character buffer beginning at r to the area beginning at l.

RETURNS

Nothing.

NAME
    p_disp - free allocated datum

SYNOPSIS
```
    VOID p_disp(p)
        TEXT *p;
```

FUNCTION
    p_disp returns the space pointed at by p to the heap, from which it must have been earlier allocated by a call to p_new.

RETURNS
    Nothing. If p is nil, p_disp aborts. If p points to an item that was never allocated from the heap, the C runtime will complain and abort.

NAME

p_fget - get data from text file

SYNOPSIS

```
VOID p_fget(pfile, fmt, arg1, ...)
    TEXT *pfile, *fmt;
    ...;
```

FUNCTION

p_fget reads text, from the file associated with the variable at pfile, and converts fields for assignment to the variables pointed at by arg1, ... A field is converted for each character in the NUL terminated string at fmt, by the following rules:

a - causes the next argument to be taken as a pointer to character and used to assign the next character from input.

c - causes the next argument to be taken as a pointer to character and used to assign the next decimal field from input. A decimal field is the internal integer representation of a string of decimal digits, which may be preceded by an arbitrary amount of whitespace (non-printing characters) and an optional '+' or '-' sign, and which may be up to 64 characters long.

s - causes the next argument to be taken as a pointer to short and used to assign the next decimal field from input.

l - causes the next argument to be taken as a pointer to long and used to assign the next decimal field from input.

f - causes the next argument to be taken as a pointer to float and used to assign the next floating field from input. A floating field is the internal double representation of a decimal field, which may be followed by a decimal point and a fraction part, and/or by an exponent, consisting of an 'e' or 'E' and a decimal field giving the power of ten by which the integer plus fraction is to be multiplied.

d - causes the next argument to be taken as a pointer to double and used to assign the next floating field from input.

n - causes input up to and including the next end of line to be consumed.

If any other character is present, then p_fget performs a panic abort, since the Pascal translator should never generate such a call.

The companion function p_tget is called when the input contains no 'f' or 'd' format characters, in the hopes that an extensive library of floating support may not have to be loaded on a small machine.

RETURNS

Nothing.

SEE ALSO

p_bget, p_tget

NAME
p_files - the open Pascal files

SYNOPSIS
```
PFILE p_files[NFILES];
```

FUNCTION
p_files is the array of structures that control open Pascal files. It is initialized to have the file variable input open for reading text, and the file variable output open for writing text.

p_files is packaged with p_ckfd.

SEE ALSO
input, output, p_ckfd

NAME

p_fill - validate input buffer

SYNOPSIS

```
TEXT *p_fill(pfile)
    TEXT *pfile;
```

FUNCTION

p_fill is used religiously by the Pascal translator to ensure that the file associated with the variable at pfile has its next input record correctly represented in the buffer. This implements a form of "lazy input", where the actual read operation is deferred as late as possible, to reconcile the Pascal definition of input with the needs of an interactive environment. The cost is a somewhat more expensive buffer access, at least when the buffer is accessed multiple times for each record; and it means that a function call such as eof or eoln may have the side effect of causing a read, a practice eschewed by some purists.

RETURNS

p_fill returns pfile for further consumption. If a read error occurs, it aborts; otherwise the file is guaranteed not to be in the P_INVAL state, i.e. having an invalid buffer.

SEE ALSO

p_load

NAME

p_fput - put data to text file

SYNOPSIS

```
VOID p_fput(pfile, fmt, arg1, ...)
    TEXT *pfile, *fmt;
    ...;
```

FUNCTION

p_fput writes text, to the file associated with the variable at pfile, by encoding as fields of text the arguments arg1, ... A field is written for each character in the NUL terminated string at fmt, other than '#' or '.' or 'n', by the following rules:

- the next argument is taken as the width (number of characters) of the next field to be produced. If the width is not an integer greater than zero, p_fput aborts.

. - the next argument is taken as the precision (number of characters to the right of the decimal point) of the next field to be produced, which is presumably a floating number. If the precision is not an integer greater than zero, p_fput aborts.

a - causes the next argument to be taken as an integer and written as a character integer.

b - causes the next argument to be taken as a integer and written as the sequence "True" if nonzero, else as "False".

c, s - cause the next argument to be taken as a signed short and written as an optional preceding '-' sign followed by the shortest sequence of decimal digits that represents its internal value.

l - causes the next argument to be taken as a signed long and written as an optional preceding '-' sign followed by the shortest sequence of decimal digits that represents its internal value.

' - causes the next argument to be taken as a pointer to an array of characters, whose length is specified by the integer argument after that. The array is written.

f, d - cause the next argument to be taken as a double and written as a floating number in either fixed point representation, if a precision has been specified, or floating point representation. Fixed point representation consists of a leading minus sign '-' or a space, followed by the shortest decimal string that can represent the number, with precision digits to the right of the decimal point; at most 24 characters are used. Floating point representation consists of a leading minus sign '-' or a space, followed by a fraction with one digit to the left and five digits to the right of the decimal point, followed by a decimal exponent, which consists of an 'e', a '+' or '-' sign, and either two or three digits of exponent, depending upon the target machine.

n - causes a newline to be written.

If any other character is present, then p_fput performs a panic abort, since the Pascal translator should never generate such a call.

If the field to be output is shorter than the width specified, then spaces are written before the field to ensure that exactly width characters are output. Note that a field may well be longer than width characters, however.

The companion function p_tput is called when the input contains no f or d format characters, in the hope that an extensive library of floating support may not have to be loaded on a small machine.

RETURNS

Nothing.

SEE ALSO

p_bput, p_tput

NAME

p_get - perform Pascal get

SYNOPSIS

```
VOID p_get(pfile)
    TEXT *pfile;
```

FUNCTION

p_get performs the Pascal get function on the file associated with the variable at pfile. If the file is not open for reading, or is at end of file, p_get aborts. Otherwise a valid buffer is made invalid and an invalid buffer is made valid by a call to p_read.

RETURNS

Nothing.

SEE ALSO

p_put, p_read

NAME

p_load - internal validate input buffer

SYNOPSIS

```
VOID p_load(p)
    PFILE *p;
```

FUNCTION

p_load is used by the Pascal runtime to ensure that the buffer controlled by the PFILE structure at p is not in the state P_INVAL. If it is, a read is attempted into the buffer and the file is advanced to P_EOF on an end of file, to P_EOLN on a text file end of line, or to P_VALID otherwise.

p_load is packaged with p_fill.

RETURNS

Nothing. If the read can obtain only a partial record, p_load aborts.

SEE ALSO

p_fill, p_read

NAME

p_new - allocate new datum

SYNOPSIS

```
VOID p_new(pp, size)
    TEXT **pp;
    BYTES size;
```

FUNCTION

p_new allocates space on the heap for an item of size bytes, then sets the pointer at pp to point at the start of the allocated space.

RETURNS

Nothing. If no more space can be allocated, p_new aborts; otherwise the pointer at pp is set.

SEE ALSO

p_disp

NAME

p_pnam - determine Pascal permanent filename

SYNOPSIS

```
TEXT *p_pnam(pfile)
    TEXT *pfile;
```

FUNCTION

p_pnam scans the program argument array _pargs to see if the file variable at pfile is an imported file variable, i.e. to see if it is named in the program header. If it is not, then pfile is not a permanent file. Otherwise, if there is an argument on the command line (the line typed to invoke the program), that corresponds positionally to the program argument in the program header, then the command line argument is taken as the name of the permanent file. If there is no corresponding argument, then the identifier name from _pargs is used as the name of the permanent file.

All of this malarkey is designed to emulate the bizarre behavior of the original Wirth Pascal, which had no provision for opening files by name in any other fashion than by mentioning them on the command line at program invocation time.

RETURNS

p_pnam returns NULL for a temporary file, otherwise a pointer to the NUL terminated permanent filename.

SEE ALSO

_pargs, p_rset, p_rwri

NAME

p_put - perform Pascal put

SYNOPSIS

```
VOID p_put(pfile)
    TEXT *pfile;
```

FUNCTION

p_put performs the Pascal put function on the file associated with the variable at pfile by calling p_write with the address of the buffer variable and its size. The size is taken as one byte for a text file.

RETURNS

Nothing.

SEE ALSO

p_get, p_write

NAME

p_read - internal read input buffer

SYNOPSIS

```
TEXT *p_read(p)
    PFILE *p;
```

FUNCTION

p_read is used by the Pascal runtime to ensure that the buffer controlled by the PFILE structure at p corresponds to a readable file and is not in the state P_INVAL. If the file is opened for writing, p_read aborts. Otherwise, if the file is in the state P_INVAL, i.e. has an invalid buffer, p_load is called to attempt a read and determine its proper state.

p_read is packaged with p_fill.

RETURNS

p_read returns the address of the associated buffer p->p_buf.

SEE ALSO

p_fill, p_load

NAME

p_rset - open a Pascal file for reading

SYNOPSIS

```
VOID p_rset(pfile, size)
    TEXT *pfile;
    BYTES size;
```

FUNCTION

p_rset opens for reading a file to be associated with the file variable at pfile, and sets its buffer size to size bytes; a size of zero implies a text file with buffer size of one byte. The name of the file is obtained from p_pnam if possible; otherwise the name is obtained from p_unam and the file is marked as temporary. If a file is not temporary and cannot be opened, then it is taken as a zero length file.

p_rset is packaged with p_pnam and p_rwri.

RETURNS

Nothing. An entry is made in p_files for the opened file.

SEE ALSO

p_close, p_files, p_pnam, p_rwri, p_unam

NAME

p_rwri - create a Pascal file for writing

SYNOPSIS

```
VOID p_rwri(pfile, size)
    TEXT *pfile;
    BYTES size;
```

FUNCTION

p_rwri creates for writing a file to be associated with the file variable at pfile, and sets its buffer size to size bytes; a size of zero implies a text file with buffer size of one byte. The name of the file is obtained from p_pnam if possible; otherwise the name is obtained from p_unam and the file is marked as temporary. If the file cannot be created, then p_rwri aborts.

p_rwri is packaged with p_pnam and p_rset.

RETURNS

Nothing. An entry is made in p_files for the created file.

SEE ALSO

p_close, p_files, p_pnam, p_rset, p_unam

NAME

p_sand - perform set and operation

SYNOPSIS

```
TEXT *p_sand(code, l, r)
    BYTES code;
    TEXT *l, *r;
```

FUNCTION

p_sand performs a bitwise and of the character string at r into the character string at l, i.e. each result bit is set only if both l and r bits are set. If (code & 02) the l string is modified, otherwise a new string is allocated to receive the result. If (code & 01) the r string is freed after the operation. The number of bytes in each string is given by (code >> 3).

RETURNS

p_sand returns a pointer to the result string.

SEE ALSO

p_sdif, p_sor

NAME

p_scon - perform set construction operation

SYNOPSIS

```
TEXT *p_scon(code, l, r)
    BYTES code;
    BYTES l, r;
```

FUNCTION

p_scon allocates a result character string and constructs a set in it. If ((code & 03) == 0) an empty set is constructed, i.e. all bits are zero. Otherwise if ((code & 03) == 1) all bits are zero except the bit at l. Otherwise all bits are zero except the bits at l through r inclusive.

Bits are numbered from least significant to most significant within a byte, and from lowest addressed to highest addressed byte within the string. The lowest numbered bit is called bit zero. The number of bytes in the string is given by (code >> 3).

RETURNS

p_scon returns a pointer to the allocated result string.

NAME

p_scpy - perform set copy operation

SYNOPSIS

```
TEXT *p_scpy(code, l, r)
    BYTES code;
    TEXT *l, *r;
```

FUNCTION

p_scpy copies the character string at r into the character string at l. If (code & 01) the r string is freed after the operation. The number of bytes in each string is given by (code >> 3).

RETURNS

p_scpy returns l.

NAME

p_sdif - perform set difference operation

SYNOPSIS

```
TEXT *p_sdif(code, l, r)
    BYTES code;
    TEXT *l, *r;
```

FUNCTION

p_sdif performs a bitwise difference of the character string at r from the character string at l, i.e. each result bit is set only if the l bit is set and the r bit is clear. If (code & 02) the l string is modified, otherwise a new string is allocated to receive the result. If (code & 01) the r string is freed after the operation. The number of bytes in each string is given by (code >> 3).

RETURNS

p_sdif returns a pointer to the result string.

SEE ALSO

p_sand, p_sor

NAME

p_sequ - perform set equality comparison

SYNOPSIS

```
BOOL p_sequ(code, l, r)
    BYTES code;
    TEXT *l, *r;
```

FUNCTION

p_sequ performs a bitwise equality comparison of the character string at r with the character string at l. If (code & 02) the l string is freed after the operation. If (code & 01) the r string is freed after the operation. The number of bytes in each string is given by (code >> 3).

RETURNS

p_sequ returns a nonzero result if the strings are identical, otherwise zero.

SEE ALSO

p_sleq

NAME

p_sin - perform set membership test

SYNOPSIS

```
BOOL p_sin(code, l, r)
    BYTES code, l;
    TEXT *r;
```

FUNCTION

p_sin tests if the element l is present in the character string at r, i.e. if the bit at l is set in r. Bits are numbered from least significant to most significant within a byte, and from lowest to highest address byte within the string. The lowest numbered bit is called bit zero. If (code & 01) the r string is freed after the operation. The number of bytes in the string is given by (code >> 3).

RETURNS

p_sin returns a nonzero result if l is a member of r, otherwise zero.

NAME

p_sleq - perform set inclusion test

SYNOPSIS

```
BOOL p_sleq(code, l, r)
    BYTES code;
    TEXT *l, *r;
```

FUNCTION

p_sleq performs a bitwise inclusion test of the character string at r with the character string at l; r is included in l if there is no bit set in r that is not set in l. If (code & 02) the l string is freed after the operation. If (code & 01) the r string is freed after the operation. The number of bytes in each string is given by (code >> 3).

RETURNS

p_sleq returns a nonzero result if r is included in l, otherwise zero.

SEE ALSO

p_sequ

NAME

p_sor - perform set or operation

SYNOPSIS

```
TEXT *p_sor(code, l, r)
    BYTES code;
    TEXT *l, *r;
```

FUNCTION

p_sor performs a bitwise or of the character string at r with the character string at l, i.e. each result bit is set if either the l or the r bit is set. If (code & 02) the l string is modified, otherwise a new string is allocated to receive the result. If (code & 01) the r string is freed after the operation. The number of bytes in each string is given by (code >> 3).

RETURNS

p_sor returns a pointer to the result string.

SEE ALSO

p_sand, p_sdif

NAME

p_sub - check subscript bounds

SYNOPSIS

```
BYTES p_sub(val, lo, hi)
    ARGINT val, lo, hi;
```

FUNCTION

p_sub tests whether the signed integer val is algebraically lower than the lower bound lo or higher than the upper bound hi. If it is, p_sub aborts.

RETURNS

p_sub returns (val - lo).

NAME

p_tget - get nonfloating data from text file

SYNOPSIS

```
VOID p_tget(pfile, fmt, arg1, ...)
    TEXT *pfile, *fmt;
    ...;
```

FUNCTION

p_tget reads text, from the file associated with the variable at pfile, and converts fields for assignment to the variables pointed at by arg1, ... A field is converted for each character in the NUL terminated string at fmt, by the same rules as for p_fget, except that the f and d format items are not supported by p_tget. Thus, p_tget is favored by the Pascal translator in the hopes that an extensive library of floating support may not have to be loaded on a small machine.

RETURNS

Nothing.

SEE ALSO

p_bget, p_fget

NAME

p_tput - put nonfloating data to text file

SYNOPSIS

```
VOID p_tput(pfile, fmt, arg1, ...)
    TEXT *pfile, *fmt;
    ...;
```

FUNCTION

p_tput writes text, to the file associated with the variable at pfile, by encoding as fields of text the arguments arg1, ... A field is written for each character in the NUL terminated string at fmt, by the same rules as for p_fput, except that the f and d format items are not supported by p_tput. Thus, p_tput is favored by the Pascal translator in the hopes that an extensive library of floating support may not have to be loaded on a small machine.

RETURNS

Nothing.

SEE ALSO

p_bput, p_fput

NAME

p_unam - generate Pascal temporary file name

SYNOPSIS

```
TEXT *p_unam(p)
    PFILE *p;
```

FUNCTION

p_unam creates a unique name for a temporary file and writes it into an internal buffer. The name is a concatenation of the name provided by the C system interface routine uname() and a single letter 'a' through 'p' determined by the position of the PFILE entry in p_files pointed at by p.

p_unam is packaged with p_close.

RETURNS

p_unam returns a pointer to its internal buffer in which the name has been written as a NUL terminated string.

SEE ALSO

p_files

BUGS

It generates a garbage suffix character if p doesn't point inside p_files.

NAME

p_write - internal write output buffer

SYNOPSIS

```
VOID p_write(p, buf, size, name)
    PFILE *p;
    TEXT *buf;
    BYTES size;
    TEXT *name;
```

FUNCTION

p_write writes size characters, starting at buf, to the file under control of the PFILE structure at p. If the file is not open for writing, or if not all characters can be written, then p_write aborts, using name as the name of the aborting program. Otherwise a text file is left in the P_WROTE state if the last character written is not a newline; it is left in the P_WRITE state for any other write.

p_write is packaged with p_put.

RETURNS

Nothing.

SEE ALSO

p_put

NAME

page - put page delimiter

SYNOPSIS

```
VOID page(pfile)
    TEXT *pfile;
```

FUNCTION

page puts a formfeed (ASCII FF, or 014) to the text file associated with the variable at pfile. If a partial line has been output, the line is terminated with a newline before the formfeed is put out.

RETURNS

Nothing. If the file is not opened for writing text, page aborts.