II. Standard Pascal Subprograms

NAME

Builtins - the standard Pascal subprograms

FUNCTION

The functions and procedures documented here are a standard part of all Pascal implementations (or should be). They are documented, as closely as possible, in the style of external procedures that might be provided were they not builtin. This often requires multiple, or arm-waving, declarations because many of the facilities are exceptions to the normal typing rules of Pascal -- which is why they have to be builtin to the language to begin with.

Nevertheless, the builtin subprograms are documented in the same style as optional library subprograms, such as those in Section III or in the C Programmers' Manual. The major difference is that these need not be, and should never be, declared before use in a Pascal program.

NAME

abs - compute absolute value

SYNOPSIS

```
function abs(i: integer): integer;
```

OR

```
function abs(x: real): real;
```

FUNCTION

abs computes the absolute value of an expression of type integer or real.

No check is made against overflow.

RETURNS

The return value of abs has the type and absolute value of its argument.

NAME

arctan - arctangent

SYNOPSIS

```
function arctan(x: real): real;
```

FUNCTION

arctan computes the angle in radians whose tangent is x, to full double precision. It works by folding x into the interval [0, 1], then interpolating from an eight entry table, using the sum of tangents formula and a fifth-order telescoped Taylor series approximation.

RETURNS

arctan returns the nearest internal representation to arctan x, expressed as a double floating value in the interval (-pi/2, pi/2).

EXAMPLE

To find the phase angle of a vector:

```
theta := arctan(y / x) * 180 / pi;
```

NAME

chr - convert integer to char

SYNOPSIS

```
function chr(i: integer): char;
```

FUNCTION

chr finds the value of type char having the ordinal position specified by its argument, which may be any expression of type integer.

RETURNS

In this implementation, chr simply returns the value of its argument, with its type changed from integer to char; no range checking is performed.

NAME

cos - cosine in radians

SYNOPSIS

```
function cos(x: real): real;
```

FUNCTION

cos computes the cosine of x, expressed in radians, to full double precision. It works by scaling x in quadrants, then computing the appropriate sin or cos of an angle in the first half-quadrant, using a sixth-order telescoped Taylor series approximation. If the magnitude of x is too large to contain a fractional quadrant part, the value is 1.

RETURNS

cos returns the nearest internal representation to cos x, expressed as a double floating value.

EXAMPLE

To rotate a vector through the angle theta:

```
xnew := xold * cos(theta) - yold * sin(theta);
ynew := xold * sin(theta) + yold * cos(theta);
```

SEE ALSO

sin

NAME

dispose - deallocate a variable created with new

SYNOPSIS

```
procedure dispose(var p: pointertype);
```

OR

```
procedure dispose(var p: pointertype;
    c1: ordinaltype; ...; cn: ordinaltype);
```

FUNCTION

dispose releases the storage used for a variable previously allocated with new. The actual parameter corresponding to p must be a variable of a pointer type, whose value uniquely identifies the variable to be discarded.

The second and succeeding arguments, if given, are used in some implementations for the disposal of a record variable containing variants that were specified in the call to new that created the variable. The values specified here should match the ones originally given in the call to new. In this implementation, however, these arguments are ignored.

EXAMPLE

To pop a stack:

```
p := ptop;
ptop := p^.next;
dispose(p);
```

SEE ALSO

new

NAME

eof - test for end of input file

SYNOPSIS

```
function eof(var f: filetype): boolean;
```

OR

```
function eof: boolean;
```

FUNCTION

eof tests whether end-of-file has been encountered on the file associated with f. The actual parameter corresponding to f must be a variable of file type. If the parameter is omitted, then the call is assumed to refer to the pre-defined textfile input.

RETURNS

eof returns true if the file is opened for output, or if it has been positioned past end-of-file. eof returns false if the file is opened for input and there is more information to be read. If f is not associated with an open file, eof aborts.

SEE ALSO

eoln

NAME

eoln - test for end of input line

SYNOPSIS

```
function eoln(var f: filetype): boolean;
```

OR

```
function eoln: boolean;
```

FUNCTION

eoln tests whether end-of-line has been encountered on the textfile associated with f. The actual parameter corresponding to f must be a variable of type text. If the parameter is omitted, the call is assumed to refer to the pre-defined textfile input.

RETURNS

eoln returns true if the textfile 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 textfile is opened for reading and is not positioned at end-of-line. Otherwise it aborts.

SEE ALSO

eof

NAME

exp - exponential

SYNOPSIS

```
function exp(x: real): real;
```

FUNCTION

exp computes the exponential of x to full double precision. It works by expressing x/ln 2 as an integer plus a fraction in the interval (-1/2, 1/2]. The exponential of this fraction is approximated by a ratio of two seventh-order polynomials.

RETURNS

exp returns the nearest internal representation to exp x, expressed as a double floating value. If the result is too large to be properly represented, a range error condition is raised; if that is inhibited, the largest representable value is returned.

EXAMPLE

```
sinh := (exp(x) - exp(-x)) / 2;
```

SEE ALSO

ln

NAME

get - fill a file buffer from the file

SYNOPSIS

```
procedure get(var f: filetype);
```

FUNCTION

get reads the next value in sequence from the file indicated by f, and places it in the associated buffer variable f^. The actual parameter corresponding to f must be a variable of a file type that has been initialized for input with reset.

Note that an implicit get occurs when a file is reset.

EXAMPLE

To copy a textfile, using the pre-defined variables input and output:

```
program testget(input, output);
begin
    while (not eof) do
        begin
        if (eoln) then
            writeln
        else
            begin
            output^ := input^;
            put(output)
            end;
        get(input)
        end;
end.
```

To copy a binary file of integers from inf to outf:

```
program binary(inf, outf);

var
    inf, outf: file of integer;

begin
    reset(inf);
    rewrite(outf);
    while (not eof(inf)) do
        begin
        outf^ := inf^;
        put(outf);
        get(inf)
        end;
end.
```

SEE ALSO

put, read, readln, reset

NAME

ln - natural logarithm

SYNOPSIS

```
function ln(x: real) : real;
```

FUNCTION

ln computes the natural log of x to full double precision. It works by expressing x as a fraction in the interval [1/2, 1), times an integer power of two. The logarithm of the fraction is approximated by a sixth-order telescoped series approximation.

RETURNS

ln returns the nearest internal representation to ln x, expressed as a double floating value. If x is negative or zero, a domain error condition is raised.

EXAMPLE

```
arcsinh := ln(x + sqrt(x * x + 1));
```

SEE ALSO

exp

NAME

new - allocate an instance of a variable

SYNOPSIS

```
procedure new(var p: pointertype);
```

OR

```
procedure new(var p: pointertype;
    c1: ordinaltype; ...; cn: ordinaltype);
```

FUNCTION

new allocates storage for a variable of the type to which p is a pointer, and assigns p a value that uniquely identifies the newly-created variable. The actual parameter corresponding to p must be a variable of a pointer type.

The second and succeeding arguments, if given, are used in some implementations to specify constants in the tag fields of successive variant parts in the newly-allocated record. If these arguments appear, then the domain type of p must be a record type, and the types of values given must match in order the tag field types of its variants. Values may be given for a leading sequence of tag fields, or for all of them. In this implementation, however, these arguments are ignored.

EXAMPLE

To grow a stack:

```
new(p);
p^.next := ptop;
ptop := p;
```

SEE ALSO

dispose

NAME

odd - test low-order bit of an integer

SYNOPSIS

function odd(i: integer): boolean;

FUNCTION

odd tests whether its argument is exactly divisible by two. The argument may be any expression of type integer.

RETURNS

odd returns true if the value of its argument is odd; otherwise, it returns false.

NAME

ord - find position in type of an ordinal expression

SYNOPSIS

```
function ord(n: ordinaltype): integer;
```

FUNCTION

ord finds the position of the value of its argument within the series of values determined by the type of the argument. The argument may be an expression of any ordinal type.

An integer quantity has an ordinal position equal to its own value. The first value of an enumerated type has the ordinal position zero.

RETURNS

ord returns the ordinal position of the value of its argument.

EXAMPLE

Given the enumerated type:

```
feel = (flaccid, soft, bouncy, firm, petrous);
```

each of the following expressions would have the value 2:

```
i := ord(bouncy);
i := ord(abs(-2));
```

NAME

page - output a page break to a textfile

SYNOPSIS

```
procedure page(var f: text);
OR
procedure page;
```

FUNCTION

page writes a form-feed character (ASCII FF or octal 014) to the file indicated by f. The actual parameter corresponding to f must be a variable of type text that has been initialized for output. If the parameter is omitted, then the call is assumed to refer to the pre-defined file output.

If the last previous output to f was not terminated by an end-of-line, page writes an end-of-line to f before writing the form-feed.

EXAMPLE

```
page;
writeln('Title ', title);
```

NAME

pred - return preceding value in ordinal type

SYNOPSIS

```
function pred(n: ordinaltype): ordinaltype;
```

FUNCTION

pred obtains the value of type ordinaltype with an ordinal position one less than that of the argument n; ordinaltype may be any ordinal type.

RETURNS

In this implementation, pred simply returns the preceding value from the range of values determined by the type of its argument. No range checking is performed.

SEE ALSO

succ

NAME

put - flush a file buffer to the file

SYNOPSIS

```
procedure put(var f: filetype);
```

FUNCTION

put outputs to the file indicated by f the current value of its associated buffer variable f^. The actual parameter corresponding to f must be a variable of a file type that has been initialized for output with rewrite.

EXAMPLE

To copy a textfile, using the pre-defined variables input and output:

```
program testput(input, output);
begin
    while (not eof) do
        begin
        if (eoln) then
            writeln
        else
            begin
            output^ := input^;
            put(output)
            end;
        get(input)
        end;
end.
```

To copy a binary file of integers from inf to outf:

```
program binary(inf, outf);

var
    inf, outf: file of integer;

begin
    reset(inf);
    rewrite(outf);
    while (not eof(inf)) do
        begin
        outf^ := inf^;
        put(outf);
        get(inf)
        end;
end.
```

SEE ALSO

get, rewrite, write, writeln

NAME

read - read a list of variables from a file

SYNOPSIS

```
procedure read(var f: filetype;
    var v1: valtype; ...; var vn: valtype);
OR
procedure read(var f: text;
    var v1: valtype; ...; var vn: valtype);
OR
procedure read(var v1: valtype; ...; var vn: valtype);
```

FUNCTION

read inputs a series of values from the file indicated by f and assigns them to the variables specified in its call. The actual parameter corresponding to f must be a variable of a file type that has been initialized for input with reset. It is followed by one or more variables of the types explained below. read operates in three different ways, depending on the type (and presence or absence) of f.

If f is given and is not of type text, then the call must specify a series of variables of the same type as the values in the file, which are assigned consecutive values read from the file.

If f is given and is of type text, then each variable specified in the call may be of type real, char, integer, or subrange of char or integer. In this case, read will try to input a value appropriate to each variable, as follows:

1) if the variable is of type char, a single character is read into it.

2) if the variable is of type integer, then read skips leading whitespace and tries to scan an integer constant (as defined in Section I under Syntax). If it does so, the converted value of the constant is assigned to the variable. Otherwise an error is reported.

3) if the variable is of type real, then read tries to scan a real constant in the same way, and assigns the converted result to the variable.

If f is not specified, then the call is presumed to refer to the predefined textfile input, and read operates as for any other textfile.

SEE ALSO

readln

NAME

readln - read a line of input from a textfile

SYNOPSIS

```
procedure readln(var f: text;
    var v1: valtype; ...; var vn: valtype);
OR
procedure readln(var v1: valtype; ...; var vn: valtype);
```

FUNCTION

readln reads a line of input from the file indicated by f. The actual parameter corresponding to f must be a variable of type text that has been initialized for input with reset. If the parameter is omitted, then the call is assumed to refer to the pre-defined file input.

A readln call need not specify any variables (vi). If it does, they will be assigned values from the line read, as described under read.

readln always leaves f positioned at the character following the first end-of-line it encounters, or at end-of-file if no such character exists.

SEE ALSO

read

NAME

reset - initialize a file for input

SYNOPSIS

```
procedure reset(var f: filetype);
```

FUNCTION

reset opens for input the file indicated by f, and reads the first value in the file into its associated buffer variable f^. The actual parameter corresponding to f must be a variable of a file type.

SEE ALSO

rewrite

NAME

rewrite - initialize a file for output

SYNOPSIS

```
procedure rewrite(var f: filetype);
```

FUNCTION

rewrite truncates to zero length the file indicated by f, then opens it for output. The actual parameter corresponding to f must be a variable of a file type.

SEE ALSO

reset

NAME

round - round real to integer

SYNOPSIS

```
function round(x: real): integer;
```

FUNCTION

round increases the absolute value of its argument by 0.5, then finds the nearest integer closer to zero than the modified argument. The actual parameter corresponding to x may be any expression of type real.

RETURNS

round returns the integer corresponding to its argument, rounded.

SEE ALSO

trunc

NAME

sin - sine in radians

SYNOPSIS

```
function sin(x: real): real;
```

FUNCTION

sin computes the sine of x, expressed in radians, to full double precision. It works by scaling x in quadrants, then computing the appropriate sin or cos of an angle in the first half-quadrant, using a sixth-order telescoped Taylor series approximation. If the magnitude of x is too large to contain a fractional quadrant part, the value of sin is 0.

RETURNS

sin returns the nearest internal representation to sin x, expressed as a double floating value.

EXAMPLE

To rotate a vector through the angle theta:

```
xnew := xold * cos(theta) - yold * sin(theta);
ynew := xold * sin(theta) + yold * cos(theta);
```

SEE ALSO

cos

NAME

sqr - square an argument

SYNOPSIS

```
function sqr(i: integer): integer;
```

OR

```
function sqr(x: real): real;
```

FUNCTION

sqr finds the square of its argument, which may be an expression of type integer or real.

No check is made against overflow.

RETURNS

sqr returns the square of its argument, which naturally shares its type.

NAME

sqrt - real square root

SYNOPSIS

```
function sqrt(x: real): real;
```

FUNCTION

sqrt computes the square root of x to full double precision. It works by expressing x as a fraction in the interval [1/2, 1), times an integer power of two. The square root of the fraction is obtained by three iterations of Newton's method, using a quadratic approximation as a starting value.

RETURNS

sqrt returns the nearest internal representation to sqrt x, expressed as a double floating value. If x is negative, a domain error condition is raised.

EXAMPLE

To find the magnitude of a vector:

```
mag := sqrt(x * x + y * y);
```

SEE ALSO

exp

NAME

succ - return succeeding value in ordinal type

SYNOPSIS

function succ(n: ordinaltype): ordinaltype;

FUNCTION

succ obtains the value of type ordinaltype with an ordinal position one greater than that of the argument n; ordinaltype may be any ordinal type.

RETURNS

In this implementation, succ simply returns the succeeding value from the range of values determined by the type of its argument. No range checking is performed.

SEE ALSO

pred

NAME
trunc - truncate real to integer

SYNOPSIS
```
function trunc(x: real): integer;
```

FUNCTION
trunc converts a real number to the nearest integer closer to zero than the original number. The actual parameter corresponding to x may be any expression of type real.

RETURNS
trunc returns the integer corresponding to its argument, truncated.

SEE ALSO
round

NAME

write - write a list of variables to a file

SYNOPSIS

```
procedure write(var f: filetype;
    e1: valtype; ...; en: valtype);
OR
procedure write(var f: text;
    e1: writeparamtype; ...; en: writeparamtype);
OR
procedure write(e1: writeparamtype; ...; en: writeparamtype);
```

FUNCTION

write outputs to the file indicated by f the values of the expressions specified in its call. The actual parameter corresponding to f must be a variable of a file type that has been initialized for output with rewrite. It is followed by one or more expressions of the types explained below. write operates in three different ways, depending on the type (and presence or absence) of f.

If f is present and is not of type text, then the remaining arguments to the call must specify a series of expressions of the same type as f, which will be written with no conversion to the file.

If f is present and is of type text, then the remaining arguments to the call each specify an output descriptor of the form:

<writexpr> [: <width> [: <frac>]]

where <writexpr> is an expression of type real, boolean, char, integer, or subrange of char or integer; a <writexpr> may also be of a string type. <width> and <frac> are optional, and are integer expressions giving the total width of the field output, and the number of fractional digits it is to contain. The latter expression affects the output only of real numbers. Each argument is converted to an ASCII string before output, as follows:

1) an argument of type string, char, or subrange of char is output literally.

2) an argument of type boolean is output as the string "True", if non-zero, else as "False".

3) an argument of type integer or subrange of integer is output as a minus sign '-', if necessary, followed by the shortest decimal string representing its value.

4) if <frac> is given, a real argument is output as a leading minus sign '-' or a space, followed by the shortest decimal string representing its value, with <frac> digits to the right of the decimal point. At most 24 characters are output.

5) if <frac> is omitted, a real argument is output as 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, consisting of an 'e', a '+' or '-' sign, and two or three digits of exponent, depending on the target machine.

If <width> is not given, the field output will be exactly as wide as the converted argument. If <width> is given, and the converted argument has fewer than <width> characters, spaces are written before it to assure that exactly <width> characters are output. A converted argument longer than <width> is not truncated, however.

If f is not given, then the call is presumed to refer to the pre-defined textfile output, and write will operate as for any other textfile.

SEE ALSO
writeln

NAME

writeln - write a line of output to a textfile

SYNOPSIS

```
procedure writeln(var f: text;
    e1: writeparamtype; ...; en: writeparamtype);
OR
procedure writeln(e1: writeparamtype; ...; en: writeparamtype);
```

FUNCTION

writeln writes a line of output to the file indicated by f. The actual parameter corresponding to f must be a variable of type text that has been initialized for output with rewrite. If the parameter is omitted, then the call is assumed to refer to the pre-defined file output.

A writeln call need not specify any expressions (ei). If it does, their values will be output to the line written, as described under write.

writeln always terminates its output with a single end-of-line character.

SEE ALSO

write