NAME

scanf1 -- formatted input scanner

SYNOPSIS

```
int scanf1([-j[,input-string]],control-string,arg1,arg2,...)
char *input-string;
char *control-string;
```

DESCRIPTION

Scanf1 is patterned after the interface existing for the portable library routine scanf. It was developed to perform most of the features offered by scanf without incurring the penalty of scanf's size (approximately 7000 bytes). The size of scanf1 is about 1650 bytes.

Scanf1 is designed to read either from terminals or strings. On reads from terminals, scanf provides its own buffer. Terminal reads in excess of 100 characters may cause errors.

Scanf1 reads characters, interprets them according to a format and stores the results in its arguments. It expects as arguments:

1. An optional input-string, indicating the source of the input characters; if omitted the standard input is read.
2. A control-string described below.
3. A set of arguments, each of which must be a pointer, indicating where the converted input should be stored.

The integer j must be in the range of 4>j>0. If (j&1) is not equal to zero, the optional input string is to be specified. If (j&2) is not equal to zero, indirection is specified. See the description for format specification "i" below.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. Blanks, tabs or newlines which are ignored.
2. Conversion specifications, consisting of the character %, an optional assignment suppressing character *, and optional numerical field width, and a conversion character.

A conversion specification is used to direct the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by the * character. An input field is defined as a string of non-space characters; it extends either to the next space character or until the field width, if specified, is exhausted.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. Pointers, rather than variable names, are re-

quired by the "call-by-value" semantics of the C language. The following conversion characters are legal:

- d indicates that a decimal integer is expected in the input stream; the corresponding argument should be an integer pointer.
- o indicates that an octal integer is expected in the input stream; the corresponding argument should be an integer pointer.
- s indicates that a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating "\0", which will be added. The input field is terminated by a space character or a newline.
- a indicates that a character string of non-space, non-slash, non-exclamation point characters is expected at this point. Otherwise it is handled as for "s" above.
- r indicates that all internal pointers are to be reset. From the terminal this will force a read.
- i indicates that the next argument in the call to scanf is to be taken as the address of a new argument list. All converted inputs are stored as directed by this argument list. There is no return to the original argument list.
- c indicates that a single character is expected; the corresponding argument should be a character pointer; the next input character is placed at the indicated spot. The normal skip over space characters is suppressed in this case; to read the next non-space character use %1s.
- [indicates a string not to be delimited by space characters. The left bracket is followed by a set of characters and a right bracket; the characters between the brackets define a set of characters making up the string. If the first character is not circumflex (^), the input field is all characters until the first character not in the set between the brackets; if the first character after the left bracket is ^, the input field is all characters until the first character which is in the remaining set of characters between the brackets. The corresponding argument must point to a character array. Right bracket may be escaped within brackets by preceding it with back slash.

For example, the call:

```
int i;int j;char name[50];
scanf1("%d%o%a",&i,&j,name);
```

with the input line

```
77 77 test/
```

will assign to i the value of 77, to j the value of octal 77, and name will contain "test\O". The subsequent call

scanf1("%1s",name)

will move the string "/\O" into the array name.

Care should be exercised when reading from the terminal. If a format is specified such that it successfully matches to the end of the last string read, another read will be made from the terminal. This might cause the program to go to sleep on the terminal. The conversion character "a" is designed to make this problem easier to avoid from the SCCS shell.

Scanf1 returns as its value the number of successfully matched and assigned input items. This can be used to decide how many input items were found. On end of file, -1 is returned; note that this is different from 0, which means that the next input character does not match what you called for in the control string. Scanf1, if given a first argument of -1, will scan a string in memory given as the second argument. It differs from scanf in that the switching of input streams from a terminal to a string causes the pointers to the terminal stream to be lost. If a subsequent read is made to the terminal it should be reinitialized with the conversion character r. All scans from a string are automatically reinitialized.

LIBRARY

/lib/lib1.a

SEE ALSO

scanf(1S)

RESTRICTIONS

Used only prior to SC6.