NAME
ascanf – argument-line scanner similar in syntax to scanf

SYNOPSIS
cc -c -I/home/5156/rabsonlib ...
cc -o ... -L/home/5156/rabsonlib -lrabson

#include <rabsonlib.h>

int ascanf(argc, argv, format_line, address1, ....)
char **argv, *format_line;
TYPE *address1,....;

DESCRIPTION
This is a medium-duty, easy-to-use argument-line parser for C programs: the idea is to avoid a huge and ugly switch/case construct in the main() function. ascanf (3) understands most UNIX argument-line conventions and makes it easy to extract numeric and string arguments and flags.

SUMMARY
A detailed description follows this summary.
%d, %s, etc. Stuff the corresponding pointer with the given object.
# Return the address of a string; same precedence as %s.
-c|num If argument -c is specified, OR the return value with num.
-c&num If argument -c is specified, AND the return value with num.
-cF Expect "-c value", "-c=value", "-c:value", or "-cvalue", where F is a scanf format or #.
|num or &num may be combined with this. Multiple formats are allowed. For example,
"-c1%d%s" will read arguments "-c 42 foo" or "-c=17 foo" and set the 1 flag.
E%s%s If any errors are encountered in parsing argv, do not print the error messages on stderr;
instead, stuff the first corresponding string with a description of the error (which will include a single ‘%s’) and the second string with the offending argument.
+ As FIRST character ONLY: do not complain about arguments in argv that match no formats. Instead, set all elements of argv that were read normally to NULL, and leave only those that matched no formats as they were.
=num Set the default flag value to num; this is useful in conjunction with & flags, which can unset bits in the flag.

MORE DETAILED DESCRIPTION
argc and argv are the usual arguments of main(); format_line is a string similar to the format string of scanf (3); I describe it more fully below. The remaining arguments are addresses to be stuffed with arguments that have been read.

The format string consists of a sequence of modified scanf formats separated by single spaces. The most common format string is %C where C is one of s, d, c, f, lf, x, or o specifying the type of argument expected in the command line. As with scanf (3), any argument matching the n th such format will be stuffed into the n th user-supplied variable, the addresses of which are specified by address1 , ... above. Example:

ascanf(argc, argv, "%s %d", string, &i);

If a given word in the argument line (specified by argv) could match more than one of the formats in format_line, ascanf chooses the least general: for example, %o is less general than %d, so if both are specified in format_line, a single argument of ‘7’ will match %o. %s is the most general format type, so a given argument will match it only if it fails to match anything else. Format modifiers, e.g., %2.2s, are supported. Since %s is not always convenient, ascanf (3) also supports a new format, # (pound sign). Pound
sign matches a string just as does %s, but the corresponding address must be a (char **): the (char *) to which it points will be stuffed with the address of any matching string in argv.

ascanf (3) provides a very simple way to find out what flags have been specified. Normally, the programmer will associate with each legal flag an arbitrary power of two. In the format line, he or she will then specify flags in the form "-C|num", where C is the flag expected and num is the associated value. ascanf() will return the bitwise OR of all flags that match. It should be noted that ascanf() returns -1 if there was an error, in which case the flags are lost. Example:

```c
#define CFLAG 0x1
#define HFLAG 0x4
sprintf(format, "-c%d -h%d", CFLAG, HFLAG);
if((flags=ascanf(argc, argv, format))==1)
(error)
```

As indicated in the short summary above, it is also possible to AND values when flags are found. One may also specify that a flag has some value:

```c
ascanf(argc, argv, ",-g%s|4", string);
```

The above example looks for a -g flag with a value, which may be specified in the argument line as ",-g value", ",-g:value", ",-g=value", or ",-g=value". The \|4 above is optional, and the |num and value parts of the string format may come in either order.

If there is a minus sign (" - ") sitting by itself in the argument line given by argv, it is not read, but the next argument will not be interpreted as a switch, even if it begins with a minus sign. Note that a number beginning with a minus sign (e.g., -3.14159) will be interpreted as a number, not a switch, so long as there are no numeric switches (e.g., -9) in the format. If there are numeric switches, the user must supply the minus sign (" - ") sitting by itself before a negative number she wishes not to be interpreted as a switch.

The summary section above shows how to override ascanf’s normal method of dealing with errors.

**ORIGINALLY WRITTEN 1/88**

**BUGS**

Not all argument lines can be parsed with ascanf, and while it is a big improvement over continually-growing switch/case’s or if/else’s, it is not as neat as the table-run-function approach, which could also give the user more specific and useful error messages. The latter, however, requires more work than most programmers are willing to put into a short program, for which ascanf is more appropriate.

There is currently no provision for a switch that both ands the flag with some number and ors it with some other, e.g., -x&0xffbf|0x1000.

SunOS 5.7 Last change: 16 August 1993 2