NAME

regexpr, compile, step, advance – regular expression compile and match routines

SYNOPSIS

cc [*flag* ...] *file* ... **–lgen** [*library* ...]

#include <regexpr.h>

char ∗**compile(const char** ∗*instring*, **char** ∗*expbuf*, **char** ∗*endbuf***);**

int step(const char ∗*string*, **char** ∗*expbuf***);**

int advance(const char ∗*string*, **char** ∗*expbuf***);**

extern char ∗**loc1,** ∗**loc2,** ∗**locs;**
extern int nbra, regerrno, reglength;
extern char ∗**braslist[],** ∗**braelist[];**

MT-LEVEL

MT-Safe

DESCRIPTION

These routines are used to compile regular expressions and match the compiled expressions against lines. The regular expressions compiled are in the form used by **ed**(1).

The parameter *instring* is a null-terminated string representing the regular expression.

The parameter *expbuf* points to the place where the compiled regular expression is to be placed. If *expbuf* is **NULL**, **compile( )** uses **malloc**(3C) to allocate the space for the compiled regular expression. If an error occurs, this space is freed. It is the user's responsibility to free unneeded space after the compiled regular expression is no longer needed.

The parameter *endbuf* is one more than the highest address where the compiled regular expression may be placed. This argument is ignored if *expbuf* is **NULL**. If the compiled expression cannot fit in (*endbuf*–*expbuf*) bytes, **compile( )** returns **NULL** and **regerrno** (see below) is set to 50.

The parameter *string* is a pointer to a string of characters to be checked for a match. This string should be null-terminated.

The parameter *expbuf* is the compiled regular expression obtained by a call of the function **compile( )**.

The function **step( )** returns non-zero if the given string matches the regular expression, and zero if the expressions do not match. If there is a match, two external character pointers are set as a side effect to the call to **step( ) .** The variables set in **step( )** are **loc1** and **loc2**. **loc1** is a pointer to the first character that matched the regular expression. The variable **loc2** points to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire line, **loc1** points to the first character of *string* and **loc2** points to the null at the end of *string*.

The purpose of **step( )** is to step through the *string* argument until a match is found or until the end of *string* is reached. If the regular expression begins with ˆ, **step( )** tries to match the regular expression at the beginning of the string only.

The **advance( )** function is similar to **step( )**; but, it only sets the variable **loc2** and always restricts matches to the beginning of the string.

If one is looking for successive matches in the same string of characters, **locs** should be set equal to **loc2**, and **step( )** should be called with *string* equal to **loc2**. **locs** is used by commands like **ed** and **sed** so that global substitutions like **s/y*//g** do not loop forever, and is **NULL** by default.

The external variable **nbra** is used to determine the number of subexpressions in the compiled regular expression. **braslist** and **braelist** are arrays of character pointers that point to the start and end of the **nbra** subexpressions in the matched string. For example, after calling **step( )** or **advance( )** with string **sabcdefg** and regular expression **\(abcdef\)**, **braslist[0]** will point at **a** and **braelist[0]** will point at **g**. These arrays are used by commands like **ed** and **sed** for substitute replacement patterns that contain the *n* notation for subexpressions.

Note that it is not necessary to use the external variables **regerrno**, **nbra**, **loc1**, **loc2 locs**, **braelist**, and **braslist** if one is only checking whether or not a string matches a regular expression.

EXAMPLES

The following is similar to the regular expression code from **grep**:

```
#include <regexpr.h>

. . .
if(compile(*argv, (char *)0, (char *)0) == (char *)0)
        regerr(regerrno);
. . .
if (step(linebuf, expbuf))
        succeed( );
```

RETURN VALUES

If **compile( )** succeeds, it returns a non-**NULL** pointer whose value depends on *expbuf*. If *expbuf* is non-**NULL**, **compile( )** returns a pointer to the byte after the last byte in the compiled regular expression. The length of the compiled regular expression is stored in **reglength**. Otherwise, **compile( )** returns a pointer to the space allocated by **malloc**.

The functions **step( )** and **advance( )** return non-zero if the given string matches the regular expression, and zero if the expressions do not match.

ERRORS

If an error is detected when compiling the regular expression, a **NULL** pointer is returned from **compile( )** and **regerrno** is set to one of the non-zero error numbers indicated below:

ERROR	MEANING
11	Range endpoint too large.
16	Bad number.
25	''\digit'' out of range.
36	Illegal or missing delimiter.
41	No remembered search string.
42	**\(~\)** imbalance.
43	Too many **\(**.

44	More than 2 numbers given in \{˜\}.
45	} expected after \.
46	First number exceeds second in \{˜\}.
49	**[ ]** imbalance.
50	Regular expression overflow.

ENVIRONMENT

If any of the **LC_**∗ variables (**LC_CTYPE, LC_MESSAGES, LC_TIME, LC_COLLATE, LC_NUMERIC,** and **LC_MONETARY**) (see **environ**(5)) are not set in the environment, the operational behavior of **tar** for each corresponding locale category is determined by the value of the **LANG** environment variable. If **LC_ALL** is set, its contents are used to override both the **LANG** and the other **LC_**∗ variables. If none of the above variables is set in the environment, the "C" (U.S. style) locale determines how **tar** behaves.

LC_CTYPE
: Determines how **tar** handles characters. When **LC_CTYPE** is set to a valid value, **tar** can display and handle text and filenames containing valid characters for that locale. **tar** can display and handle Extended Unix code (EUC) characters where any individual character can be 1, 2, or 3 bytes wide. **tar** can also handle EUC characters of 1, 2, or more column widths. In the "C" locale, only characters from ISO 8859-1 are valid.

SEE ALSO

ed(1), **grep**(1), **sed**(1), **malloc**(3C), **regexp**(5)

NOTES

When compiling multi-thread applications, the **_REENTRANT** flag must be defined on the compile line. This flag should only be used in multi-thread applications.