PROLOG

This manual page is part of the POSIX Programmer's Manual. The Cygwin implementation of this interface may differ (consult the corresponding Cygwin manual page for details of Cygwin behavior), or the interface may not be implemented on Cygwin.

NAME

regcomp, regerror, regexec, regfree — regular expression matching

SYNOPSIS

```
#include <regex.h>

int regcomp(regex_t *restrict preg, const char *restrict pattern,
   int cflags);
size_t regerror(int errcode, const regex_t *restrict preg,
   char *restrict errbuf, size_t errbuf_size);
int regexec(const regex_t *restrict preg, const char *restrict string,
   size_t nmatch, regmatch_t pmatch[restrict], int eflags);
void regfree(regex_t *preg);
```

DESCRIPTION

These functions interpret *basic* and *extended* regular expressions as described in the Base Definitions volume of POSIX.1-2008, *Chapter 9*, *Regular Expressions*.

The **regex_t** structure is defined in <*regex.h*> and contains at least the following member:

Member Type	Member Name	Description
size_t	*re_nsub*	Number of parenthesized subexpressions.

The **regmatch_t** structure is defined in <*regex.h*> and contains at least the following members:

Member Type	Member Name	Description
regoff_t	*rm_so*	Byte offset from start of *string* to start of substring.
regoff_t	*rm_eo*	Byte offset from start of *string* of the first character after the end of substring.

The *regcomp*() function shall compile the regular expression contained in the string pointed to by the *pattern* argument and place the results in the structure pointed to by *preg*. The *cflags* argument is the bitwise-inclusive OR of zero or more of the following flags, which are defined in the <*regex.h*> header:

REG_EXTENDED
: Use Extended Regular Expressions.

REG_ICASE
: Ignore case in match (see the Base Definitions volume of POSIX.1-2008, *Chapter 9*, *Regular Expressions*).

REG_NOSUB
: Report only success/fail in *regexec*().

REG_NEWLINE
: Change the handling of <newline> characters, as described in the text.

The default regular expression type for *pattern* is a Basic Regular Expression. The application can specify Extended Regular Expressions using the REG_EXTENDED *cflags* flag.

If the REG_NOSUB flag was not set in *cflags*, then *regcomp*() shall set *re_nsub* to the number of parenthesized subexpressions (delimited by "\(\)" in basic regular expressions or "()" in extended regular expressions) found in *pattern*.

The *regexec*() function compares the null-terminated string specified by *string* with the compiled regular expression *preg* initialized by a previous call to *regcomp*(). If it finds a match, *regexec*() shall return 0; otherwise, it shall return non-zero indicating either no match or an error. The *eflags* argument is the bitwise-inclusive OR of zero or more of the following flags, which are defined in the <*regex.h*> header:

REG_NOTBOL
: The first character of the string pointed to by *string* is not the beginning of the line. Therefore, the <circumflex> character (**'^'**), when taken as a special character, shall not match the beginning of *string*.

REG_NOTEOL
: The last character of the string pointed to by *string* is not the end of the line. Therefore, the <dollar-sign> (**'$'**), when taken as a special character, shall not match the end of *string*.

If *nmatch* is 0 or REG_NOSUB was set in the *cflags* argument to *regcomp*(), then *regexec*() shall ignore the *pmatch* argument. Otherwise, the application shall ensure that the *pmatch* argument points to an array with at least *nmatch* elements, and *regexec*() shall fill in the elements of that array with offsets of the substrings of *string* that correspond to the parenthesized subexpressions of *pattern*: *pmatch*[*i*].*rm_so* shall be the byte offset of the beginning and *pmatch*[*i*].*rm_eo* shall be one greater than the byte offset of the end of substring *i*. (Subexpression *i* begins at the *i*th matched open parenthesis, counting from 1.) Offsets in *pmatch*[0] identify the substring that corresponds to the entire regular expression. Unused elements of *pmatch* up to *pmatch*[*nmatch*−1] shall be filled with −1. If there are more than *nmatch* subexpressions in *pattern* (*pattern* itself counts as a subexpression), then *regexec*() shall still do the match, but shall record only the first *nmatch* substrings.

When matching a basic or extended regular expression, any given parenthesized subexpression of *pattern* might participate in the match of several different substrings of *string*, or it might not match any substring even though the pattern as a whole did match. The following rules shall be used to determine which substrings to report in *pmatch* when matching regular expressions:

1. If subexpression *i* in a regular expression is not contained within another subexpression, and it participated in the match several times, then the byte offsets in *pmatch*[*i*] shall delimit the last such match.
2. If subexpression *i* is not contained within another subexpression, and it did not participate in an otherwise successful match, the byte offsets in *pmatch*[*i*] shall be −1. A subexpression does not participate in the match when:

   '*' or **"\{\}"** appears immediately after the subexpression in a basic regular expression, or **'*'**, **'?'**, or **"{ }"** appears immediately after the subexpression in an extended regular expression, and the subexpression did not match (matched 0 times)

   or:

   > **'|'** is used in an extended regular expression to select this subexpression or another, and the other subexpression matched.
3. If subexpression *i* is contained within another subexpression *j*, and *i* is not contained within any other subexpression that is contained within *j*, and a match of subexpression *j* is reported in *pmatch*[*j*], then the match or non-match of subexpression *i* reported in *pmatch*[*i*] shall be as described in 1. and 2. above, but within the substring reported in *pmatch*[*j*] rather than the whole string. The offsets in *pmatch*[*i*] are still relative to the start of *string*.
4. If subexpression *i* is contained in subexpression *j*, and the byte offsets in *pmatch*[*j*] are −1, then the pointers in *pmatch*[*i*] shall also be −1.
5. If subexpression *i* matched a zero-length string, then both byte offsets in *pmatch*[*i*] shall be the byte offset of the character or null terminator immediately following the zero-length string.

If, when *regexec*() is called, the locale is different from when the regular expression was compiled, the result is undefined.

If REG_NEWLINE is not set in *cflags*, then a <newline> in *pattern* or *string* shall be treated as an ordinary character. If REG_NEWLINE is set, then <newline> shall be treated as an ordinary character except as follows:

1. A <newline> in *string* shall not be matched by a <period> outside a bracket expression or by any form of a non-matching list (see the Base Definitions volume of POSIX.1-2008, *Chapter 9*, *Regular*

Expressions).

2. A <circumflex> (**'^'**) in *pattern*, when used to specify expression anchoring (see the Base Definitions volume of POSIX.1-2008, *Section 9.3.8*, *BRE Expression Anchoring*), shall match the zero-length string immediately after a <newline> in *string*, regardless of the setting of REG_NOTBOL.
3. A <dollar-sign> (**'$'**) in *pattern*, when used to specify expression anchoring, shall match the zero-length string immediately before a <newline> in *string*, regardless of the setting of REG_NOTEOL.

The *regfree*() function frees any memory allocated by *regcomp*() associated with *preg*.

The following constants are defined as the minimum set of error return values, although other errors listed as implementation extensions in *<regex.h>* are possible:

REG_BADBR Content of **"\{\}"** invalid: not a number, number too large, more than two numbers, first larger than second.

REG_BADPAT Invalid regular expression.

REG_BADRPT **'?'**, **'*'**, or **'+'** not preceded by valid regular expression.

REG_EBRACE **"\{\}"** imbalance.

REG_EBRACK **"[]"** imbalance.

REG_ECOLLATE Invalid collating element referenced.

REG_ECTYPE Invalid character class type referenced.

REG_EESCAPE Trailing <backslash> character in pattern.

REG_EPAREN **"\(\)"** or **"()"** imbalance.

REG_ERANGE Invalid endpoint in range expression.

REG_ESPACE Out of memory.

REG_ESUBREG Number in **"\digit"** invalid or in error.

REG_NOMATCH *regexec*() failed to match.

If more than one error occurs in processing a function call, any one of the possible constants may be returned, as the order of detection is unspecified.

The *regerror*() function provides a mapping from error codes returned by *regcomp*() and *regexec*() to unspecified printable strings. It generates a string corresponding to the value of the *errcode* argument, which the application shall ensure is the last non-zero value returned by *regcomp*() or *regexec*() with the given value of *preg*. If *errcode* is not such a value, the content of the generated string is unspecified.

If *preg* is a null pointer, but *errcode* is a value returned by a previous call to *regexec*() or *regcomp*(), the *regerror*() still generates an error string corresponding to the value of *errcode*, but it might not be as detailed under some implementations.

If the *errbuf_size* argument is not 0, *regerror*() shall place the generated string into the buffer of size *errbuf_size* bytes pointed to by *errbuf*. If the string (including the terminating null) cannot fit in the buffer, *regerror*() shall truncate the string and null-terminate the result.

If *errbuf_size* is 0, *regerror*() shall ignore the *errbuf* argument, and return the size of the buffer needed to hold the generated string.

If the *preg* argument to *regexec*() or *regfree*() is not a compiled regular expression returned by *regcomp*(), the result is undefined. A *preg* is no longer treated as a compiled regular expression after it is given to

regfree().

RETURN VALUE

Upon successful completion, the *regcomp*() function shall return 0. Otherwise, it shall return an integer value indicating an error as described in <*regex.h*>, and the content of *preg* is undefined. If a code is returned, the interpretation shall be as given in <*regex.h*>.

If *regcomp*() detects an invalid RE, it may return REG_BADPAT, or it may return one of the error codes that more precisely describes the error.

Upon successful completion, the *regexec*() function shall return 0. Otherwise, it shall return REG_NOMATCH to indicate no match.

Upon successful completion, the *regerror*() function shall return the number of bytes needed to hold the entire generated string, including the null termination. If the return value is greater than *errbuf_size*, the string returned in the buffer pointed to by *errbuf* has been truncated.

The *regfree*() function shall not return a value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

```
#include <regex.h>

/*
 * Match string against the extended regular expression in
 * pattern, treating errors as no match.
 *
 * Return 1 for match, 0 for no match.
 */

int
match(const char *string, char *pattern)
{
    int    status;
    regex_t    re;

    if (regcomp(&re, pattern, REG_EXTENDED|REG_NOSUB) != 0) {
        return(0);      /* Report error. */
    }
    status = regexec(&re, string, (size_t) 0, NULL, 0);
    regfree(&re);
    if (status != 0) {
        return(0);      /* Report error. */
    }
    return(1);
}
```

The following demonstrates how the REG_NOTBOL flag could be used with *regexec*() to find all substrings in a line that match a pattern supplied by a user. (For simplicity of the example, very little error checking is done.)

```
(void) regcomp (&re, pattern, 0);
/* This call to regexec() finds the first match on the line. */
error = regexec (&re, &buffer[0], 1, &pm, 0);
while (error == 0) {  /* While matches found. */
    /* Substring found between pm.rm_so and pm.rm_eo. */
    /* This call to regexec() finds the next match. */
    error = regexec (&re, buffer + pm.rm_eo, 1, &pm, REG_NOTBOL);
```

```
}
```

APPLICATION USAGE

An application could use:

```
regerror(code,preg,(char *)NULL,(size_t)0)
```

to find out how big a buffer is needed for the generated string, *malloc*() a buffer to hold the string, and then call *regerror*() again to get the string. Alternatively, it could allocate a fixed, static buffer that is big enough to hold most strings, and then use *malloc*() to allocate a larger buffer if it finds that this is too small.

To match a pattern as described in the Shell and Utilities volume of POSIX.1-2008, *Section 2.13*, *Pattern Matching Notation*, use the *fnmatch*() function.

RATIONALE

The *regexec*() function must fill in all *nmatch* elements of *pmatch*, where *nmatch* and *pmatch* are supplied by the application, even if some elements of *pmatch* do not correspond to subexpressions in *pattern*. The application developer should note that there is probably no reason for using a value of *nmatch* that is larger than *preg*->*re_nsub*+1.

The REG_NEWLINE flag supports a use of RE matching that is needed in some applications like text editors. In such applications, the user supplies an RE asking the application to find a line that matches the given expression. An anchor in such an RE anchors at the beginning or end of any line. Such an application can pass a sequence of <newline>-separated lines to *regexec*() as a single long string and specify REG_NEWLINE to *regcomp*() to get the desired behavior. The application must ensure that there are no explicit <newline> characters in *pattern* if it wants to ensure that any match occurs entirely within a single line.

The REG_NEWLINE flag affects the behavior of *regexec*(), but it is in the *cflags* parameter to *regcomp*() to allow flexibility of implementation. Some implementations will want to generate the same compiled RE in *regcomp*() regardless of the setting of REG_NEWLINE and have *regexec*() handle anchors differently based on the setting of the flag. Other implementations will generate different compiled REs based on the REG_NEWLINE.

The REG_ICASE flag supports the operations taken by the *grep* **−i** option and the historical implementations of *ex* and *vi*. Including this flag will make it easier for application code to be written that does the same thing as these utilities.

The substrings reported in *pmatch*[ ] are defined using offsets from the start of the string rather than pointers. This allows type-safe access to both constant and non-constant strings.

The type **regoff_t** is used for the elements of *pmatch*[ ] to ensure that the application can represent large arrays in memory (important for an application conforming to the Shell and Utilities volume of POSIX.1-2008).

The 1992 edition of this standard required **regoff_t** to be at least as wide as **off_t**, to facilitate future extensions in which the string to be searched is taken from a file. However, these future extensions have not appeared. The requirement rules out popular implementations with 32-bit **regoff_t** and 64-bit **off_t**, so it has been removed.

The standard developers rejected the inclusion of a *regsub*() function that would be used to do substitutions for a matched RE. While such a routine would be useful to some applications, its utility would be much more limited than the matching function described here. Both RE parsing and substitution are possible to implement without support other than that required by the ISO C standard, but matching is much more complex than substituting. The only difficult part of substitution, given the information supplied by *regexec*(), is finding the next character in a string when there can be multi-byte characters. That is a much larger issue, and one that needs a more general solution.

The *errno* variable has not been used for error returns to avoid filling the *errno* name space for this feature.

The interface is defined so that the matched substrings *rm_sp* and *rm_ep* are in a separate **regmatch_t** structure instead of in **regex_t**. This allows a single compiled RE to be used simultaneously in several

contexts; in *main*() and a signal handler, perhaps, or in multiple threads of lightweight processes. (The *preg* argument to *regexec*() is declared with type **const**, so the implementation is not permitted to use the structure to store intermediate results.) It also allows an application to request an arbitrary number of substrings from an RE. The number of subexpressions in the RE is reported in *re_nsub* in *preg*. With this change to *regexec*(), consideration was given to dropping the REG_NOSUB flag since the user can now specify this with a zero *nmatch* argument to *regexec*(). However, keeping REG_NOSUB allows an implementation to use a different (perhaps more efficient) algorithm if it knows in *regcomp*() that no subexpressions need be reported. The implementation is only required to fill in *pmatch* if *nmatch* is not zero and if REG_NOSUB is not specified. Note that the **size_t** type, as defined in the ISO C standard, is unsigned, so the description of *regexec*() does not need to address negative values of *nmatch*.

REG_NOTBOL was added to allow an application to do repeated searches for the same pattern in a line. If the pattern contains a <circumflex> character that should match the beginning of a line, then the pattern should only match when matched against the beginning of the line. Without the REG_NOTBOL flag, the application could rewrite the expression for subsequent matches, but in the general case this would require parsing the expression. The need for REG_NOTEOL is not as clear; it was added for symmetry.

The addition of the *regerror*() function addresses the historical need for conforming application programs to have access to error information more than ''Function failed to compile/match your RE for unknown reasons''.

This interface provides for two different methods of dealing with error conditions. The specific error codes (REG_EBRACE, for example), defined in *<regex.h>*, allow an application to recover from an error if it is so able. Many applications, especially those that use patterns supplied by a user, will not try to deal with specific error cases, but will just use *regerror*() to obtain a human-readable error message to present to the user.

The *regerror*() function uses a scheme similar to *confstr*() to deal with the problem of allocating memory to hold the generated string. The scheme used by *strerror*() in the ISO C standard was considered unacceptable since it creates difficulties for multi-threaded applications.

The *preg* argument is provided to *regerror*() to allow an implementation to generate a more descriptive message than would be possible with *errcode* alone. An implementation might, for example, save the character offset of the offending character of the pattern in a field of *preg*, and then include that in the generated message string. The implementation may also ignore *preg*.

A REG_FILENAME flag was considered, but omitted. This flag caused *regexec*() to match patterns as described in the Shell and Utilities volume of POSIX.1-2008, *Section 2.13*, *Pattern Matching Notation* instead of REs. This service is now provided by the *fnmatch*() function.

Notice that there is a difference in philosophy between the ISO POSIX-2: 1993 standard and POSIX.1-2008 in how to handle a ''bad'' regular expression. The ISO POSIX-2: 1993 standard says that many bad constructs ''produce undefined results'', or that ''the interpretation is undefined''. POSIX.1-2008, however, says that the interpretation of such REs is unspecified. The term ''undefined'' means that the action by the application is an error, of similar severity to passing a bad pointer to a function.

The *regcomp*() and *regexec*() functions are required to accept any null-terminated string as the *pattern* argument. If the meaning of the string is ''undefined'', the behavior of the function is ''unspecified''. POSIX.1-2008 does not specify how the functions will interpret the pattern; they might return error codes, or they might do pattern matching in some completely unexpected way, but they should not do something like abort the process.

FUTURE DIRECTIONS

None.

SEE ALSO

fnmatch( ), *glob*( )

The Base Definitions volume of POSIX.1-2008, *Chapter 9*, *Regular Expressions*, **<regex.h>**, **<sys_types.h>**

The Shell and Utilities volume of POSIX.1-2008, *Section 2.13*, *Pattern Matching Notation*

COPYRIGHT