regcomp() – Compile regular expression

Last Updated: 2021-04-12

Standards

Standards / Extensions	C or C++	Dependencies
XPG4 XPG4.2 Single UNIX Specification, Version 3 z/OS® UNIX	both	

Format

```
#include <regex.h>

int regcomp(regex_t *_restrict_ preg, const char *_restrict_ pattern, int cflags);
```

General description

Compiles the regular expression specified by *pattern* into an executable string of op-codes.

preg is a pointer to a compiled regular expression.

pattern is a pointer to a character string defining a source regular expression (described below).

cflags is a bit flag defining configurable attributes of compilation process:

REG_EXTENDED
Support extended regular expressions.

REG_ICASE
Ignore case in match.

REG_NEWLINE
Eliminate any special significance to the newline character.

REG_NOSUB
Report only success or fail in regexec(), that is, verify the syntax of a regular expression. If this flag is set, the regcomp() function sets *re_nsub* to the number of parenthesized sub-expressions found in *pattern*. Otherwise, a sub-expression results in an error.

The regcomp() function under z/OS XL C/C++ will use the definition of characters according to the current LC_SYNTAX category. The characters, [,], {, }, |, ^, and $, have varying code points in different encoded character sets.

Regular expressions

The functions regcomp(), regerror(), regexec(), and regfree() use regular expressions in a similar way to the UNIX awk, ed, grep, and egrep commands.

The simplest form of regular expression is a string of characters with no special meaning. The following characters do have special meaning; they are used to form extended regular expressions:

Symbol
Description

.
The period symbol matches any one character except the terminal newline character.

[*character-character*]
The hyphen symbol, within square brackets, means “through”. It fills in the intervening characters according to the current collating sequence. For example, [a-z] can be equivalent to [abc…xyz] or, with a different collating sequence, it can be equivalent to [aAbBcC…xXyYzZ].

[*string*]
A string within square brackets specifies any of the characters in *string*. Thus [abc], if compared to other strings, would match any that contained a, b, or c.
No assumptions are made at compile time about the actual characters contained in the range.

{*m*} {*m*,} {*m*,*u*}
Integer values enclosed in {} indicate the number of times to apply the preceding regular expression. *m* is the minimum number, and *u* is the maximum number. *u* must not be greater than RE_DUP_MAX (see limits.h — Standard values for limits on resources).
If you specify only *m*, it indicates the exact number of times to apply the regular expression. {*m*,} is equivalent to {*m*,*u*}. They both match m or more occurrences of the expression.

*****
The asterisk symbol indicates 0 or more of any characters. For example, [a*e] is equivalent to any of the following: 99ae9, aaaaae, a999e99.

$
The dollar symbol matches the end of the string. (Use \n to match a newline character.)

***character*+**

The plus symbol specifies one or more occurrences of a character. Thus, smith+ern is equivalent to, for example, smithhhern.

[^*string*]

The caret symbol, when inside square brackets, negates the characters within the square brackets. Thus [^abc], if compared to other strings, would *fail* to match any that contains even one a, b, or c.

(*expression*)$*n*

Stores the value matched by the enclosed regular expression in the $(n+1)^{th}$ *ret* parameter. Ten enclosed regular expressions are allowed. Assignments are made unconditionally.

(*expression*)

Groups a sub-expression allowing an operator, such as *, +, or [].], to work on the sub-expression enclosed in parentheses. For example, (a*(cb+)*)$0.

Note:

1. Do *not* use multibyte characters.
2. You can use the] (right square bracket) alone within a pair of square brackets, but only if it immediately follows either the opening left square bracket or if it immediately follows [^. For example: []–] matches the] and – characters.
3. All the preceding symbols are *special*. You precede them with \ to use the symbol itself. For example, a\.e is equivalent to a.e.
4. You can use the – (hyphen) by itself, but only if it is the first or last character in the expression. For example, the expression []--0] matches either the] or else the characters – through 0. Otherwise, use \-.

Returned value

If successful, regcomp() returns 0.

If unsuccessful, regcomp() returns nonzero, and the content of *preg* is undefined.

Example

CELEBR07

```
/* CELEBR07

   This example compiles an extended regular expression.

 */
#include <regex.h>
#include <locale.h>
#include <stdio.h>
#include <stdlib.h>

main() {
    regex_t    preg;
    char       *string = "a simple string";
    char       *pattern = ".*(simple).*";
    int        rc;

    if ((rc = regcomp(&preg, pattern, REG_EXTENDED)) != 0) {
       printf("regcomp() failed, returning nonzero (%d)", rc);
       exit(1);
    }
}
```

Related information

- regex.h – Regular expression functions
- regerror() – Return error message
- regexec() – Execute compiled regular expression
- regfree() – Free memory for regular expression

Parent topic:

→ Library functions

Previous
regcmp() – Compile regular expression

Next
regerror() – Return error message