## NAME

regcomp, regcomplit, regcompnl, regexec, regsub, rregexec, rregsub, regerror – regular expression

## SYNOPSIS

```
#include <u.h>
#include <libc.h>
#include <regexp.h>

Reprog  *regcomp(char *exp)

Reprog  *regcomplit(char *exp)

Reprog  *regcompnl(char *exp)

int  regexec(Reprog *prog, char *string, Resub *match, int msize)

void regsub(char *source, char *dest, int dlen, Resub *match, int msize)

int  rregexec(Reprog *prog, Rune *string, Resub *match, int msize)

void rregsub(Rune *source, Rune *dest, int dlen, Resub *match, int msize)

void regerror(char *msg)
```

## DESCRIPTION

*Regcomp* compiles a regular expression and returns a pointer to the generated description. The space is allocated by *malloc*(2) and may be released by *free*. Regular expressions are exactly as in *regexp*(6).

*Regcomplit* is like *regcomp* except that all characters are treated literally. *Regcompnl* is like *regcomp* except that the `.` metacharacter matches all characters, including newlines.

*Regexec* matches a null-terminated *string* against the compiled regular expression in *prog*. If it matches, *regexec* returns 1 and fills in the array *match* with character pointers to the substrings of *string* that correspond to the parenthesized subexpressions of *exp*: `match[`*i*`].sp` points to the beginning and `match[`*i*`].ep` points just beyond the end of the *i*th substring. (Subexpression *i* begins at the *i*th left parenthesis, counting from 1.) Pointers in `match[0]` pick out the substring that corresponds to the whole regular expression. Unused elements of *match* are filled with zeros. Matches involving `*`, `+`, and `?` are extended as far as possible. The number of array elements in *match* is given by *msize*. The structure of elements of *match* is:

```
typedef struct {
        union {
           char *sp;
           Rune *rsp;
        };
        union {
           char *ep; Rune *rep;
        }; } Resub;
```

If `match[0].sp` is nonzero on entry, *regexec* starts matching at that point within *string*. If `match[0].ep` is nonzero on entry, the last character matched is the one preceding that point.

*Regsub* places in *dest* a substitution instance of *source* in the context of the last *regexec* performed using *match*. Each instance of `\`*n*, where *n* is a digit, is replaced by the string delimited by `match[`*n*`].sp` and `match[`*n*`].ep`. Each instance of `&` is replaced by the string delimited by `match[0].sp` and `match[0].ep`. The substitution will always be null terminated and trimmed to fit into dlen bytes.

*Regerror*, called whenever an error is detected in *regcomp*, writes the string *msg* on the standard error file and exits. *Regerror* can be replaced to perform special error processing. If the user supplied *regerror* returns rather than exits, *regcomp* will return 0.

*Rregexec* and *rregsub* are variants of *regexec* and *regsub* that use strings of `Runes` instead of strings of `chars`. With these routines, the *rsp* and *rep* fields of the *match* array elements should be used.

**SOURCE**

`/sys/src/libregexp`

**SEE ALSO**

*grep*(1)

**DIAGNOSTICS**

*Regcomp* returns 0 for an illegal expression or other failure. *Regexec* returns 0 if *string* is not matched.

**BUGS**

There is no way to specify or match a NUL character; NULs terminate patterns and strings.

## NAME

regexp – regular expression notation

## DESCRIPTION

A *regular expression* specifies a set of strings of characters. A member of this set of strings is said to be *matched* by the regular expression. In many applications a delimiter character, commonly /, bounds a regular expression. In the following specification for regular expressions the word 'character' means any character (rune) but newline.

The syntax for a regular expression `e0` is

```
e3:  literal | charclass | '.' | '^' | '$' | '(' e0 ')'

e2:  e3
  |  e2 REP

REP: '*' | '+' | '?'

e1:  e2
  |  e1 e2

e0:  e1
  |  e0 '|' e1
```

A `literal` is any non-metacharacter, or a metacharacter (one of `.*+?[]()|\^$`), or the delimiter preceded by `\`.

A `charclass` is a nonempty string *s* bracketed `[` *s* `]` (or `[^`*s* `]`); it matches any character in (or not in) *s*. A negated character class never matches newline. A substring *a*−*b*, with *a* and *b* in ascending order, stands for the inclusive range of characters between *a* and *b*. In *s*, the metacharacters `-`, `]`, an initial `^`, and the regular expression delimiter must be preceded by a `\`; other metacharacters have no special meaning and may appear unescaped.

A `.` matches any character.

A `^` matches the beginning of a line; `$` matches the end of the line.

The `REP` operators match zero or more (`*`), one or more (`+`), zero or one (`?`), instances respectively of the preceding regular expression `e2`.

A concatenated regular expression, `e1 e2`, matches a match to `e1` followed by a match to `e2`.

An alternative regular expression, `e0 | e1`, matches either a match to `e0` or a match to `e1`.

A match to any part of a regular expression extends as far as possible without preventing a match to the remainder of the regular expression.

## SEE ALSO

*awk*(1), *ed*(1), *grep*(1), *sam*(1), *sed*(1), *regexp*(2)