NAME

regcmp, regex − compile and execute regular expressions

SYNOPSIS

char *regcmp(string1[,string2,...],0);
char *string1, *string2, ...;

char *regex(re,subject[,ret0,...]);
char *re, *subject, *ret0, ...;

DESCRIPTION

Regcmp compiles a regular expression and returns a pointer to the compiled form. The regular expression is the concatenation of *string1*, *string2*, etc. *Alloc(III)* is used to create space for the vector. It is the user's responsibility to free unneeded space so allocated. A zero return from *regcmp* indicates an incorrect argument. *Regcmp(I)* has been written to generally preclude the need for this routine at execution time.

Regex executes a compiled pattern (*re*) against the *subject* string. Additional arguments are passed to receive values back. *Re gex* returns zero on failure or a pointer to the next unmatched character on success. A global character pointer *loc1* points to where the match began. *Regcmp* and *regex* were mostly borrowed from the editor, *ed* (I); however, the syntax and semantics have been changed slightly.

symbols	*meaning*
[ ] *.^	These symbols retain their current meaning.
$	Matches the end of the string; '\n' matches the newline.
−	Within brackets the minus means *through.* For example, [a−z] is equivalent to [abcd **...** xyz]. The '−' can appear as itself only if used as the last or first character. For example, the character class expression [ ] −] matches the characters '] ' and '−'.
+	A regular expression followed by '+' means *one or more times.* F or example, [0−9]+ is equivalent to [0−9][0−9]*
{m} *{m,}* *{m,u}*	Integer values enclosed in { } indicate the number of times the preceeding regular expression is to be applied. *m* is the minimum number and *u* is a number , less than 256, which is the maximum. If only *m* is present,e.g. {m}, *m* indicates the exact number of times the regular expression is to be applied. {m,} is analogous to {m,infinity}. The plus ('+') and star ('*') operations are equivalent to {1,} and {0,} respectively.
(. . .)$*n*	The value of the enclosed regular expression is to be returned. The matched string will be copied into the area pointed to by the *retn* argument (see the examples below). At present, at most ten enclosed regular expressions are allowed. *Regex* makes its assignments unconditionally.
(. . .)	Parentheses are used for grouping. An operator, e.g. *,+,{ }, can work on a single character or a regular expression enclosed in parenthesis. For example, (a*(cb+)*)$0.

Of necessity, all the above defined symbols are special. They must, therefore, be escaped to be used as themselves.

Example 1:

```
char *cursor, *newcursor, *ptr;
...
newcursor = regex((ptr=regcmp("^\n",0)),cursor);
free(ptr);
```

This example will match a leading newline in the subject string pointed at by cursor.

Example 2:

```
char ret0[9];
char *newcursor, *name;
...
name = regcmp("([A−Za−z][A−za−z0−9]{0,7})$0",0);
newcursor = regex(name,"123Testing321",ret0);
```

This example will match through the string "Testing3" and will return the address of the character after the last matched character (i.e., *newcursor* will point to the substring "21"). The string "Testing3" will be copied to the character array *ret0*.

Example 3:

```
#include "file.i"
char *string, *newcursor;
...
newcursor = regex(name,string);
```

This example applies a precompiled regular expression in *file.i* against *string* (see *regcmp*(I)).

Regcmp and *regex* are kept in the **−lPW** library.

SEE ALSO

regcmp(I), ed(I), alloc(III)

BUGS

The user program may run out of memory if regcmp() is called iteratively without freeing the vectors no longer required. The following user-supplied replacement for *alloc(III)* re-uses the same vector saving time and space.

```
/* user's program */
        ...
alloc(n) {
static int rebuf[256];
        return &rebuf;
}
free(ptr)
char *ptr;
{}
```