An example of using regular expressions in C

This page was last changed on **Fri Jul 10 2015**.

This example program uses the Unix regular expression library.

The compiled program takes two arguments. The first is a regular expression. The second is the text to match. When run, it matches the regular expression against the text until no more matches can be found. It then prints the matching string and up to nine parenthesized expressions.

The printout uses the Perl terminology of **$&** and **$1**.

The regular expression library itself is documented under **regex**. The format of the regular expressions is described under **re_format**.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <regex.h>

/* The following is the size of a buffer to contain any error messages
   encountered when the regular expression is compiled. */

#define MAX_ERROR_MSG 0x1000

/* Compile the regular expression described by "regex_text" into
   "r". */

static int compile_regex (regex_t * r, const char * regex_text)
{
    int status = regcomp (r, regex_text, REG_EXTENDED|REG_NEWLINE);
    if (status != 0) {
        char error_message[MAX_ERROR_MSG];
        regerror (status, r, error_message, MAX_ERROR_MSG);
        printf ("Regex error compiling '%s': %s\n",
                regex_text, error_message);
        return 1;
    }
    return 0;
}

/*
  Match the string in "to_match" against the compiled regular
  expression in "r".
 */

static int match_regex (regex_t * r, const char * to_match)
{
    /* "P" is a pointer into the string which points to the end of the
       previous match. */
    const char * p = to_match;
    /* "N_matches" is the maximum number of matches allowed. */
    const int n_matches = 10;
    /* "M" contains the matches found. */
    regmatch_t m[n_matches];

    while (1) {
        int i = 0;
        int nomatch = regexec (r, p, n_matches, m, 0);
```

```
            if (nomatch) {
                printf ("No more matches.\n");
                return nomatch;
            }
            for (i = 0; i < n_matches; i++) {
                int start;
                int finish;
                if (m[i].rm_so == -1) {
                    break;
                }
                start = m[i].rm_so + (p - to_match);
                finish = m[i].rm_eo + (p - to_match);
                if (i == 0) {
                    printf ("$& is ");
                }
                else {
                    printf ("$%d is ", i);
                }
                printf ("'%.*s' (bytes %d:%d)\n", (finish - start),
                        to_match + start, start, finish);
            }
            p += m[0].rm_eo;
        }
        return 0;
    }

    int main(int argc, char ** argv)
    {
        regex_t r;
        const char * regex_text;
        const char * find_text;
        if (argc != 3) {
            regex_text = "([[:digit:]]+)[^[:digit:]]+([[:digit:]]+)";
            find_text = "This 1 is nice 2 so 33 for 4254";
        }
        else {
            regex_text = argv[1];
            find_text = argv[2];
        }
        printf ("Trying to find '%s' in '%s'\n", regex_text, find_text);
        compile_regex(& r, regex_text);
        match_regex(& r, find_text);
        regfree (& r);
        return 0;
    }
```

(download)

When run without arguments, this produces

```
Trying to find '([[:digit:]]+)[^[:digit:]]+([[:digit:]]+)' in 'This 1 is nice 2 so 33 for 4254'
$& is '1 is nice 2' (bytes 5:16)
$1 is '1' (bytes 5:6)
$2 is '2' (bytes 15:16)
$& is '33 for 4254' (bytes 20:31)
$1 is '33' (bytes 20:22)
$2 is '4254' (bytes 27:31)
No more matches.
```

LeMoDa top page | Regular expressions | C | Examples | Illustrated