Regular expressions in C: examples?

Asked 12 years, 9 months ago Modified 2 years, 11 months ago Viewed 390k times

194

I'm after some simple examples and best practices of how to use regular expressions in ANSI C. `man regex.h` does not provide that much help.

c regex

89

Share Improve this question Follow

edited Dec 25, 2012 at 6:10
Mohammed H
6,548 14 75 122

asked Jul 5, 2009 at 23:38
Dervin Thunk
18.4k 27 116 207

7 There is no built-in support for regex in ANSI C. What regex library are you using? – Joe Jul 6, 2009 at 1:59

9 Rob Pike wrote a small regular expression string search function that accepted a very useful subset of regular expressions for the book The Practice of Programming which he and Brian Kernighan co-authored. See this discussion, A Regular Expression Matcher, by Dr. Kernighan cs.princeton.edu/courses/archive/spr09/cos333/beautiful.html – Richard Chambers Dec 14, 2014 at 23:18

5 Answers

Sorted by: Highest score (default)

Не нашли ответ? Задайте вопрос на Stack Overflow на русском. ×

260

Regular expressions actually aren't part of ANSI C. It sounds like you might be talking about the POSIX regular expression library, which comes with most (all?) *nixes. Here's an example of using POSIX regexes in C (based on this):

```
#include <regex.h>
regex_t regex;
int reti;
char msgbuf[100];

/* Compile regular expression */
reti = regcomp(&regex, "^a[[:alnum:]]", 0);
if (reti) {
    fprintf(stderr, "Could not compile regex\n");
    exit(1);
}

/* Execute regular expression */
reti = regexec(&regex, "abc", 0, NULL, 0);
```

Join Stack Overflow to find the best answer to your technical question, help others answer theirs. Sign up ×

```
    else if (reti == REG_NOMATCH) {
        puts("No match");
    }
    else {
        regerror(reti, &regex, msgbuf, sizeof(msgbuf));
        fprintf(stderr, "Regex match failed: %s\n", msgbuf);
        exit(1);
    }

    /* Free memory allocated to the pattern buffer by regcomp() */
    regfree(&regex);
```

Alternatively, you may want to check out PCRE, a library for Perl-compatible regular expressions in C. The Perl syntax is pretty much that same syntax used in Java, Python, and a number of other languages. The POSIX syntax is the syntax used by `grep` , `sed` , `vi` , etc.

Share Improve this answer Follow

edited Mar 15, 2017 at 13:37
Dmitry Egorov
9,307 3 22 39

answered Jul 6, 2009 at 2:01
Laurence Gonsalves
131k 32 230 283

7 Unless you need to avoid the dependency I second PCRE, it has some nice syntax enhancements and is very stable. At least with some older versions of Linux, the "built in" regular expression library isn't too difficult to crash given certain input strings and certain regular expressions that "almost" match or involve a lot of special characters – bdk Jul 6, 2009 at 2:16

@Laurence What's the meaning of passing 0 to regcomp? regcomp only takes four integer values 1, 2, 4 and 8 to represent 4 different modes. – lixiang Sep 21, 2013 at 7:40

2 @lixiang The last parameter to `regcomp` , `cflags` , is a bitmask. From pubs.opengroup.org/onlinepubs/009695399/functions/regcomp.html : "The cflags argument is the bitwise-inclusive OR of zero or more of the following flags...". If you OR-together zero, you'll get 0. I see that the Linux manpage for `regcomp` says "cflags may be the bitwise-or of one or more of the following", which does seem misleading. – Laurence Gonsalves Sep 22, 2013 at 18:11

2 You can extract text from matching groups with something like: `regmatch_t matches[MAX_MATCHES]; if (regexec(&exp, sz, MAX_MATCHES, matches, 0) == 0) { memcpy(buff, sz + matches[1].rm_so, matches[1].rm_eo - matches[1].rm_so); printf("group1: %s\n", buff); }` note that group matches start at 1, group 0 is the entire string. Add error checks for out of bounds, etc. – BurnsBA Feb 7, 2016 at 7:42

2 Regarding whether `regfree` is necessary after a failed `regcomp` , though it really is rather under-specified, this suggest that it shouldn't be done: redhat.com/archives/libvir-list/2013-September/msg00276.html – Daniel Jour Jun 17, 2016 at 19:50

13

It's probably not what you want, but a tool like re2c can compile POSIX(-ish) regular expressions to ANSI C. It's written as a replacement for `lex` , but this approach allows you to sacrifice flexibility and legibility for the last bit of speed, if you really need it.

Share Improve this answer Follow

answered Jul 6, 2009 at 4:20
Michiel Buddingh
5,535 19 32

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up

11

```
"^(-)?([0-9]+)((,|.)([0-9]+))?\n$"
```

Allows you to catch decimal numbers in Spanish system and international. :)

```
#include <regex.h>
#include <stdlib.h>
#include <stdio.h>
regex_t regex;
int reti;
char msgbuf[100];

int main(int argc, char const *argv[])
{
    while(1){
        fgets( msgbuf, 100, stdin );
        reti = regcomp(&regex, "^(-)?([0-9]+)((,|.)([0-9]+))?\n$",
REG_EXTENDED);
        if (reti) {
            fprintf(stderr, "Could not compile regex\n");
            exit(1);
        }

        /* Execute regular expression */
        printf("%s\n", msgbuf);
        reti = regexec(&regex, msgbuf, 0, NULL, 0);
        if (!reti) {
            puts("Match");
        }
        else if (reti == REG_NOMATCH) {
            puts("No match");
        }
        else {
            regerror(reti, &regex, msgbuf, sizeof(msgbuf));
            fprintf(stderr, "Regex match failed: %s\n", msgbuf);
            exit(1);
        }

        /* Free memory allocated to the pattern buffer by regcomp() */
        regfree(&regex);
    }

}
```

Share Improve this answer Follow

edited Nov 24, 2018 at 13:16

answered Nov 23, 2018 at 19:24
diego campos
111 1 4

2 don't you think, it s better to keep `regcomp` outside the loop?. unless it should be initialized every time it get used. – milevyo Aug 6, 2021 at 20:08

10

`man regex.h` reports there is no manual entry for regex.h, but `man 3 regex` gives you a page explaining the POSIX functions for pattern matching.
The same functions are described in The GNU C Library: Regular Expression Matching,

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up

For example, for an hypothetical program that prints which of the strings passed as argument match the pattern passed as first argument, you could use code similar to the following one.

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

void print_regerror (int errcode, size_t length, regex_t *compiled);

int
main (int argc, char *argv[])
{
  regex_t regex;
  int result;

  if (argc < 3)
    {
      // The number of passed arguments is lower than the number of
      // expected arguments.
      fputs ("Missing command line arguments\n", stderr);
      return EXIT_FAILURE;
    }
```

Frisa Lank by Child of the ’90s

```
    {
      // Any value different from 0 means it was not possible to
      // compile the regular expression, either for memory problems
      // or problems with the regular expression syntax.
      if (result == REG_ESPACE)
        fprintf (stderr, "%s\n", strerror(ENOMEM));
      else
        fputs ("Syntax error in the regular expression passed as first
argument\n", stderr);
      return EXIT_FAILURE;
    }
  for (int i = 2; i < argc; i++)
    {
      result = regexec (&regex, argv[i], 0, NULL, 0);
      if (!result)
        {
          printf ("'%s' matches the regular expression\n", argv[i]);
        }
      else if (result == REG_NOMATCH)
        {
          printf ("'%s' doesn't the regular expression\n", argv[i]);
        }
      else
        {
          // The function returned an error; print the string
          // describing it.
          // Get the size of the buffer required for the error message.
          size_t length = regerror (result, &regex, NULL, 0);
          print_regerror (result, length, &regex);
          return EXIT_FAILURE;
        }
    }

  /* Free the memory allocated from regcomp(). */
```

```
void
print_regerror (int errcode, size_t length, regex_t *compiled)
{
  char buffer[length];
  (void) regerror (errcode, compiled, buffer, length);
  fprintf(stderr, "Regex match failed: %s\n", buffer);
}
```

The last argument of `regcomp()` needs to be at least `REG_EXTENDED`, or the functions will use basic regular expressions, which means that (for example) you would need to use `a\{3\}` instead of `a{3}` used from extended regular expressions, which is probably what you expect to use.

POSIX.2 has also another function for wildcard matching: `fnmatch()`. It doesn't allow to compile the regular expression, or get the substrings matching a sub-expression, but it is very specific for checking when a filename match a wildcard (e.g. it uses the `FNM_PATHNAME` flag).

Share Improve this answer Follow

edited Mar 6, 2019 at 9:55

answered Aug 11, 2017 at 17:59
apaderno
26.5k 16 74 86

8

While the answer above is good, I recommend using **PCRE2**. This means you can literally use all the regex examples out there now and not have to translate from some ancient regex.

I made an answer for this already, but I think it can help here too..

Regex In C To Search For Credit Card Numbers

```
// YOU MUST SPECIFY THE UNIT WIDTH BEFORE THE INCLUDE OF THE pcre.h

#define PCRE2_CODE_UNIT_WIDTH 8
#include <stdio.h>
#include <string.h>
#include <pcre2.h>
#include <stdbool.h>

int main(){

bool Debug = true;
bool Found = false;
pcre2_code *re;
PCRE2_SPTR pattern;
PCRE2_SPTR subject;
int errornumber;
int i;
int rc;
PCRE2_SIZE erroroffset;
PCRE2_SIZE *ovector;
size_t subject_length;
pcre2_match_data *match_data;
```

```
pattern = (PCRE2_SPTR)RegexStr;// <<<<< This is where you pass your REGEX
subject = (PCRE2_SPTR)source;// <<<<< This is where you pass your bufer that
will be checked.
subject_length = strlen((char *)subject);




  re = pcre2_compile(
  pattern,               /* the pattern */
  PCRE2_ZERO_TERMINATED, /* indicates pattern is zero-terminated */
  0,                     /* default options */
  &errornumber,          /* for error number */
  &erroroffset,          /* for error offset */
  NULL);                 /* use default compile context */

/* Compilation failed: print the error message and exit. */
if (re == NULL)
  {
  PCRE2_UCHAR buffer[256];
  pcre2_get_error_message(errornumber, buffer, sizeof(buffer));
  printf("PCRE2 compilation failed at offset %d: %s\n",
(int)erroroffset,buffer);
  return 1;
  }


match_data = pcre2_match_data_create_from_pattern(re, NULL);

rc = pcre2_match(
  re,
  subject,              /* the subject string */
  subject_length,       /* the length of the subject */
  0,                    /* start at offset 0 in the subject */
  0,                    /* default options */
  match_data,           /* block for storing the result */
  NULL);

if (rc < 0)
  {
  switch(rc)
    {
    case PCRE2_ERROR_NOMATCH: //printf("No match\n"); //
    pcre2_match_data_free(match_data);
    pcre2_code_free(re);
    Found = 0;
    return Found;
    //  break;
    /*
    Handle other special cases if you like
    */
    default: printf("Matching error %d\n", rc); //break;
    }
  pcre2_match_data_free(match_data);   /* Release memory used for the match */
  pcre2_code_free(re);
  Found = 0;                /* data and the compiled pattern. */
  return Found;
  }


if (Debug){
ovector = pcre2_get_ovector_pointer(match_data);
printf("Match succeeded at offset %d\n", (int)ovector[0]);
if (ovector[0] > ovector[1])
  {
  printf("\\K was used in an assertion to set the match start after its end.\n"
    "From end to start the match was: %.*s\n", (int)(ovector[0] - ovector[1]),
      (char *)(subject + ovector[1]));
  printf("Run abandoned\n");
  pcre2_match_data_free(match_data);
  pcre2_code_free(re);
  return 0;
}

for (i = 0; i < rc; i++)
  {
  PCRE2_SPTR substring_start = subject + ovector[2*i];
  size_t substring_length = ovector[2*i+1] - ovector[2*i];
  printf("%2d: %.*s\n", i, (int)substring_length, (char *)substring_start);
  }
}

else{
  if(rc > 0){
    Found = true;

    }
}
pcre2_match_data_free(match_data);
pcre2_code_free(re);
return Found;

}
```

Install PCRE using:

```
wget https://ftp.pcre.org/pub/pcre/pcre2-10.31.zip
make
sudo make install
sudo ldconfig
```

Compile using :

```
gcc foo.c -lpcre2-8 -o foo
```

Check my answer for more details.

Share Improve this answer Follow

edited Jul 19, 2018 at 9:30

answered May 19, 2018 at 15:00

LUser
960 4 23 33

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up