pcre_example.c

```c
/**
   @file      pcre_example.c
   @author    Mitch Richling <http://www.mitchr.me/>
   @Copyright Copyright 1994,1998 by Mitch Richling.  All rights reserved.
   @brief     UNIX regex tools@EOL
   @Keywords  UNIX regular expressions regex perl pcre
   @Std       ISOC POSIX.2 (IEEE Std 103.2) BSD4.3

              This is an example program intended to illustrate very
              basic use of the PCRE regular expression library.  PCRE
              is NOT part of any UNIX or language standard; however,
              it is commonly found on UNIX systems today, and it has a
              compatibility mode that supports the standard
              interfaces.

              The vast irregularities found in various UNIX favors
              with regard to regular expression support can make using
              regular expressions frustrating.  It can be less
              difficult, and safer, to simply carry around a regular
              expression library with you!  PCRE is by far the most
              popular, "alternate", regular expression library
              available today.  It makes a fine choice for the budding
              UNIX programmer unwilling to explore the vulgarities of
              some operating system vendor's regular expression
              library.  If you are a C++ programmer, another good
              alternative may be found as part of the BOOST library.

              Note: This program is very similar to the
              regex_example.c example found in this directory.

   @Tested
              - Solaris 2.8
              - MacOS X.2
              - Linux (RH 7.3)
 */

#include <pcre.h>                 /* PCRE lib        NONE  */
#include <stdio.h>                /* I/O lib         C89   */
#include <stdlib.h>               /* Standard Lib    C89   */
#include <string.h>               /* Strings         C89   */

int main(int argc, char *argv[]);

int main(int argc, char *argv[]) {
  pcre *reCompiled;
  pcre_extra *pcreExtra;
  int pcreExecRet;
  int subStrVec[30];
  const char *pcreErrorStr;
  int pcreErrorOffset;
  char *aStrRegex;
  char **aLineToMatch;
  const char *psubStrMatchStr;
  int j;
  char *testStrings[] = { "This should match... hello",
                          "This could match... hello!",
                          "More than one hello.. hello",
                          "No chance of a match...",
                          NULL};


  aStrRegex = "(.*)(hello)+";
  printf("Regex to use: %s\n", aStrRegex);

  // First, the regex string must be compiled.
  reCompiled = pcre_compile(aStrRegex, 0, &pcreErrorStr, &pcreErrorOffset, NULL);

  /* OPTIONS (second argument) (||'ed together) can be:
       PCRE_ANCHORED       -- Like adding ^ at start of pattern.
       PCRE_CASELESS       -- Like m//i
       PCRE_DOLLAR_ENDONLY -- Make $ match end of string regardless of \n's
                              No Perl equivalent.
       PCRE_DOTALL         -- Makes . match newlins too.  Like m//s
       PCRE_EXTENDED       -- Like m//x
       PCRE_EXTRA          --
       PCRE_MULTILINE      -- Like m//m
       PCRE_UNGREEDY       -- Set quantifiers to be ungreedy.  Individual quantifiers
                              may be set to be greedy if they are followed by "?".
       PCRE_UTF8           -- Work with UTF8 strings.
  */
  // pcre_compile returns NULL on error, and sets pcreErrorOffset & pcreErrorStr
  if(reCompiled == NULL) {
    printf("ERROR: Could not compile '%s': %s\n", aStrRegex, pcreErrorStr);
    exit(1);
  } /* end if */

  // Optimize the regex
  pcreExtra = pcre_study(reCompiled, 0, &pcreErrorStr);

  /* pcre_study() returns NULL for both errors and when it can not optimize
     the regex.  The last argument is how one checks for errors (it is NULL
     if everything works, and points to an error string otherwise. */
  if(pcreErrorStr != NULL) {
    printf("ERROR: Could not study '%s': %s\n", aStrRegex, pcreErrorStr);
    exit(1);
  } /* end if */

  for(aLineToMatch=testStrings; *aLineToMatch != NULL; aLineToMatch++) {
    printf("String: %s\n", *aLineToMatch);
    printf("        %s\n", "0123456789012345678901234567890123456789");
    printf("        %s\n", "0         1         2         3");

    /* Try to find the regex in aLineToMatch, and report results. */
    pcreExecRet = pcre_exec(reCompiled,
                            pcreExtra,
                            *aLineToMatch,
                            strlen(*aLineToMatch),  // length of string
                            0,                      // Start looking at this point
                            0,                      // OPTIONS
                            subStrVec,
                            30);                    // Length of subStrVec

    /* pcre_exec OPTIONS (||'ed together) can be:
       PCRE_ANCHORED -- can be turned on at this time.
       PCRE_NOTBOL
       PCRE_NOTEOL
       PCRE_NOTEMPTY */

    // Report what happened in the pcre_exec call..
    //printf("pcre_exec return: %d\n", pcreExecRet);
    if(pcreExecRet < 0) { // Something bad happened..
      switch(pcreExecRet) {
      case PCRE_ERROR_NOMATCH      : printf("String did not match the pattern\n");        break;
      case PCRE_ERROR_NULL         : printf("Something was null\n");                      break;
      case PCRE_ERROR_BADOPTION    : printf("A bad option was passed\n");                 break;
      case PCRE_ERROR_BADMAGIC     : printf("Magic number bad (compiled re corrupt?)\n"); break;
      case PCRE_ERROR_UNKNOWN_NODE : printf("Something kooky in the compiled re\n");      break;
      case PCRE_ERROR_NOMEMORY     : printf("Ran out of memory\n");                       break;
      default                      : printf("Unknown error\n");                           break;
      } /* end switch */
    } else {
      printf("Result: We have a match!\n");

      // At this point, rc contains the number of substring matches found...
      if(pcreExecRet == 0) {
        printf("But too many substrings were found to fit in subStrVec!\n");
        // Set rc to the max number of substring matches possible.
        pcreExecRet = 30 / 3;
      } /* end if */

      // Do it yourself way to get the first substring match (whole pattern):
      // char subStrMatchStr[1024];
      // int i, j
      // for(j=0,i=subStrVec[0];i<subStrVec[1];i++,j++)
      //   subStrMatchStr[j] = (*aLineToMatch)[i];
      // subStrMatchStr[subStrVec[1]-subStrVec[0]] = 0;
      //printf("MATCHED SUBSTRING: '%s'\n", subStrMatchStr);

      // PCRE contains a handy function to do the above for you:
      for(j=0; j<pcreExecRet; j++) {
        pcre_get_substring(*aLineToMatch, subStrVec, pcreExecRet, j, &(psubStrMatchStr));
        printf("Match(%2d/%2d): (%2d,%2d): '%s'\n", j, pcreExecRet-1, subStrVec[j*2], subStrVec[j*2+1], psubStrMatchStr);
      } /* end for */

      // Free up the substring
      pcre_free_substring(psubStrMatchStr);
    }  /* end if/else */
    printf("\n");

  } /* end for */

  // Free up the regular expression.
  pcre_free(reCompiled);

  // Free up the EXTRA PCRE value (may be NULL at this point)
  if(pcreExtra != NULL)
```

```
    pcre_free(pcreExtra);

  // We are all done..
  return 0;

} /* end func main */
```

Generated by [GNU Enscript 1.6.5.2](http://www.codento.com/people/mtr/genscript/).