PCRE – Perl Compatible Regular Expression Library

Ami Chayun – January 1st 2007

Wouldn't it Be Nice?

```
int is_this_spam(const char *mail)
{
  if (mail =~ /stock(s)?|vi4gra|enlarge/im)
    return 1;    //death to spam
  else
    return 0;    //this can't possibly be spam
}
```

The Answer: PCRE

- Would you like to use Perl's regular expression capabilities in C your programs?
- Well, of course you would. Why else would you be coming to this lecture? (besides fighting if we should say regex or regexp)
- After this lecture you will become familiar with the capabilities of the PCRE library and what it can do for you

Table of Contents

- Short History of Regular Expression
- Why 'Perl Compatible?'
- Perl Regular Expressions Capabilities
- PCRE Overview
- PCRE C API
- Performance
- C++ API

Abbreviated History

- In the late 1960s regular expressions jumped from theoretical mathematics to the field of computer science
- The first applications commonly using regular expressions were ed, grep and later egrep
- Grep presented the * meta character, but + and ? Were not supported. It also supported capturing and other meta characters. Advanced syntax was added with new versions of egrep

Standards

- Regular expression development was not unified, for almost 20 years every application had its own flavour of regular expression.
- Tools like 'awk', 'lex' and 'sed' (and later emacs) all supported some form of regular expressions, but with fundamental differences.

POSIX Regular Expressions

- An attempt to standardize the realm of regular expressions was made in 1986 by POSIX.
- POSIX refers to two major flavors of regular expressions:
 - BRE (Basic Regular Expressions)
 - ERE (Extended Regular Expressions)
- POSIX Introduces support of different locales in both BRE and ERE

POSIX BRE

- Support dot (.), anchors (^ \$), character classes, ranges etc.
- Supports backreferences (\1 .. \9)
- Does not support alteration (|)
- Does not support the + and ? quantifiers

POSIX ERE

- Support dot (.), anchors (^ \$), character classes, ranges etc.
- Support alteration and all the quantifiers
- Does not support backreferences
- Supports locale-specific character classes like \w (character in a word)
 - Note: This was not defined in POSIX, but rather widely implemented

Regular Expression Libraries

- C has a long history of regular expression packages:
 - Henry Spencer's package (first made available at 1986 and popular until 1994)
 - GNU C Library has a POSIX compatible regular expression library (regex.h)
 - John Maddock's Regex++ (Boost regex ++) packaged with the Boost library
 - Philip Hazel's Perl 5 compatible PCRE

Other Languages

- Java included regular expression library in Java 1.4.0
- Microsoft VB 6 has simple regular expression support
- Microsoft .NET infrastructure supplies an extensive regular expression library
- All popular script languages support regular expressions. Noticeably in Perl, Ruby, Python, PHP (via pcre)

Perl Compatible

- Since the release of Perl 5, it became the de-facto standard of regular expression syntax
- Perl support all POSIX ERE syntax, as well as extensions introduced by previous utilities and languages
- Perl support Unicode and non ANSI charsets out-of-the-box

Perl Regular Expressions

- Perl's original flavor is based on the Emacs, awk and sed regular expression syntax
- Perl 2 included a complete rewrite of the regular expression engine, and evolved up to version 5
- Perl support many advanced features of regular expressions:
 - Full unicode support, unlimited number of capturing groups, lazy quantifiers, lookaround etc.

PCRE Overview

- The PCRE Library
- PCRE C example
- Using the ovector structure to access captured matches
- Perl compatible options
- Unique options to PCRE

Philip Hazel's PCRE Library

- "The PCRE library is a set of functions that implement regular expression pattern matching using the same syntax and semantics as Perl 5"
- PCRE was written for the Exim MTA.
 Version 1.0 was released on November, 1997
- Today PCRE is used by many high profile open source projects:
 - Apache web server, PHP, Postfix ...

PCRE Overview (cont.)

- The package is distributed under BSD software license
- PCRE Is available for POSIX operating systems, Mac OSX and Win32
- PCRE is written in C with a basic API, and optional wrappers
 - Most noticeably C++ and backwardcompatible POSIX regex.h API

C API Example Code

Defining our parameters

```
char *regex = "^From: ([^@]+)@([^\r]+)";
char *data = "From:
regular.expressions@example.com\r\n";
```

Compiling the expression

```
re = pcre_compile(
                  /* the pattern */
regex,
                  /* default options */
0,
                 /* for error message */
&error,
                /* for error offset */
&erroffset,
                  /* use default character table */
NULL);
if (! re)
   fprintf(stderr,
      "PCRE compilation failed at expression offset
      %d: %s\n", erroffset, error);
   return 1;
```

Executing the match

Handling match errors

```
if (rc < 0)
   switch(rc)
       case PCRE ERROR NOMATCH:
          printf("No match found in text\n");
          break;
       More cases defined...
       */
      default:
          printf("Match error %d\n", rc);
          break;
       return 1;
```

Extracting matches

```
if (rc < 3)
    printf("Match did not catch all the groups\n");
    return 1;
}
/*ovector[0]..ovector[1] are the entire matched
string*/
char *name start = data + ovector[2];
int name_length = ovector[3] - ovector[2];
char *domain_start = data + ovector[4];
int domain length = ovector[5] - ovector[4];
```

Extracting matches

```
/* Finally, print the match */
printf("Mail from: %.*s domain: %.*s\n",
name_length, name_start,
domain_length, domain_start);
return 0;
} //END main
```

Basic recipe

- Compile your expression with pcre compile
- Execute the expression with pcre exec
- Store matches in the ovector array

The Structure of ovector

 PCRE stores the match indices in an array with the following format:

 The number of captured matches is returned in the parameter rc

Ovector (cont.)

 To access group n (\$1, \$2, ...) you need:

```
int length = ovector[2*n];
const char *start =
  ovector[2*n + 1] - ovector[2*n];
```

 To allow ovector capture n groups define ovector to the size of: (n+1)*3

Compiling the Code

- To compile PCRE with GCC run: gcc pcredemo.c -lpcre -o pcredemo
- Under win32 link with pcre.lib or pcre.dll

- Obtaining PCRE:
 - From project's web site: pcre.org
 - Win32 version is distributed with GnuWin32
 - apt-get install libpcre3 libpcre3-dev

Perl Capabilities Chart

PCRE option	Usage	Perl switch
PCRE_CASELESS	case insensitive match	/i
PCRE_MULTILINE	multiple lines match	/m
PCRE_DOTALL	dot matches newlines	/s
PCRE_DOLLAR_ENDONLY	\$ matches only at end	N/A
PCRE_EXTRA	strict escape parsing	N/A
PCRE_EXTENDED	ignore whitespaces	/x
PCRE_UTF8	handles UTF8 chars	built-in
PCRE_UNGREEDY	reverses * and *?	N/A
PCRE_NO_AUTO_CAPTURE	disables matching parens	s N/A

More Options

PCRE_NEWLINE_CR

PCRE_NEWLINE_LF

PCRE_NEWLINE_CRLF

PCRE_ANCHORED

PCRE_NOTBOL

PCRE_NOTEOL

PCRE_NOTEMPTY

PCRE_NO_AUTO_CAPTURE

PCRE PARTIAL

Usage

Set the newline char to be \r

Set the newline char to be \n

Set the sequence to be \r\n

Match only at the first position

Subject is not the beginning of a line

Subject is not the beginning of a line

An empty string is not a valid match

Disable unnamed capturing parentheses

Return PCRE_ERROR_PARTIAL for a

partial match

Unique Options

PCRE option

Usage

PCRE_AUTO_CALLOUT

Automatically inserts callouts

before each item (state)

DFA Options

PCRE DFA SHORTEST

Return only the shortest match

PCRE DFA RESTART

Restart the DFA engine after a partial match

 Function callout is a unique capability to the PCRE package and allows the user to run an arbitrary function on each state of the regular expression

Performance

- Available regular expression engine types
- Expression 'study' optimization

The Engine

- Regular expressions implementations are based on two major families of algorithms:
 - NFA (Nondeterministic Finite Automaton)
 - DFA (Deterministic Finite Automaton)
- NFA also comes in a POSIX flavour

The Engine - NFA

- NFA works like DFS (Depth-First-Search)
 - Checks one possible path every time
 - Very low memory consumption
 - Fast
 - Allows capturing
- NFA is the most popular software implementation.
 - Perl uses traditional NFA

The Engine - DFA

- DFA works like BFS (Breadth-First-Search)
 - Checks all candidates at the same time,
 therefore can return all the partial matches
 - Allows unification of many regular expressions, as they can all be unified to one big state machine
 - High memory consumption (exponential)
- DFA is preferred when matching to a large number of expressions i.e: mail filters, anti-virus, IDS etc.

Alternate Engine

- PCRE also support DFA. If you wish, PCRE can execute the PCRE search with DFA engine with the pcre_dfa_exec function
- This is not Perl compatible :-)

DFA Pro / Cons

- Advantages:
 - All possible matches are found. Longest is returned (unless PCRE_DFA_SHORTEST option is specified)
 - Better support for partial matching
 - The engine doesn't backtracks
- Disadvantages:
 - Slower
 - No capturing parentheses and back references

NFA Optimization - Study

- PCRE offers an option to optimize the regular expression by running pcre_study() on a non anchored compiled regular expression.
- pcre_study creates a bitmap of possible starting characters
- This should not be confused with Perl's study, which maps the target text, rather than the expression

Study Pro / Cons

- When should you consider to study?
 - Unanchored expressions
 - pcre_study supports optimizing caseless matches (opposed to Perl study)
 - Heavily used expression
- Remember that the time and extra memory for study may not always be worth it

PCRE C++ Wrapper

- C++ Example with pcrecpp
- Differences in usage
- Supported options
- Compiling the code

C++ API

- One of the first contributions Google inc. made to the open source community was a C++ wrapper to the PCRE library. Since then it has been separately maintained.
- The C++ API supplies object oriented approach to the library, and supports std types.

C++ Example

```
#include <iostream>
#include <string>
#include <pcrecpp.h>
```

using namespace std;

Using PCRE C++

```
int main(void)
   int i;
   string s;
   pcrecpp::RE re("(\\w+):(\\d+)");
   if (re.error().length() > 0) {
      cout << "PCRE compilation failed with error: "</pre>
     << re.error() << "\n";
   if (re.PartialMatch("root:1234", &s, &i))
      cout << s << " : " << i << "\n";
```

Differences of C++ Package

- Context aware, can return: string, int, const char * etc. In capturing parentheses.
- Supports search and replace:
 - PartialMatch
 - FullMatch
 - Replace
 - GlobalReplace

Differences (cont.)

- All memory allocated internally in the object if const char * was passed.
- UTF8 support can be activated like so:

```
pcrecpp::RE_Options options;
options.set_utf8();
pcrecpp::RE re(utf8_pattern, options);
```

More PCRE C++ Notes

 Supports Perl modifiers via the RE_Options class:

```
PCRE_CASELESS
                                                    /i
                              case insensitive match
PCRE_MULTILINE
                             multiple lines match
                                                    /m
PCRE_DOTALL
                              dot matches newlines
                                                    /s
PCRE_DOLLAR_ENDONLY
                             $ matches only at end
                                                    N/A
PCRE_EXTRA
                                                     N/A
                              strict escape parsing
PCRE EXTENDED
                              ignore whitespaces
                                                     /x
PCRE_UTF8
                                                     built-in
                              handles UTF8 chars
PCRE_UNGREEDY
                                                     N/A
                              reverses * and *?
PCRE_NO_AUTO_CAPTURE
                              disables matching parens
                                                      N/A
```

 Currently does not support other PCRE flags. (easily extendible)

Compiling Code

 In addition to the pcre library, the pcrecpp headers and library are also required. With GCC:

```
g++ -lpcrecpp test_pcre_cpp.cpp -o
test_pcre_cpp
```

Under win32 link with pcrecpp.lib or pcre.dll

Obtaining PCRE:

- From the project's FTP server (under contrib dir)
- apt-get install libpcrecpp0

Bibliography

- Mastering Regular Expression (3rd edition)
- PCRE man page
- PCRE HTML man pages
- Ken Thompson -Regular expression search algorithm (1986)

Thank You.