

Parsing Sederhana dengan Regular Expression

Bagi Anda yang terbiasa bekerja dengan pemrograman atau SQL, tentunya bekerja dengan sesuatu yang melibatkan penggunaan sintaks adalah hal yang wajar. Baik sintaks dalam pemrograman ataupun *statement* SQL. Di artikel kali ini, kita akan membahas cara melakukan *parsing* sederhana secara *high level* memanfaatkan regular expression. Dengan demikian, kita bisa membuat sintaks sederhana, memeriksa apakah suatu sintaks valid atau tidak dan atau mengambil teks tertentu dalam suatu file untuk diformat lebih lanjut.

Bekerja dengan teks sebenarnya menyenangkan. Apabila Anda banyak bekerja dengan pemrograman atau SQL seperti disebutkan, banyak sekali pemrosesan teks yang terjadi pada bidang-bidang tersebut. Barangkali yang paling umum adalah bagaimana interpreter atau kompilator memeriksa apakah kode Anda benar atau tidak. Setelah itu, bagaimana interpreter atau kompilator mengenali maksud Anda mengetikkan kode program Anda. Contoh lain yang sangat umum juga adalah bagaimana database *engine* mengenali dan memahami sintaks SQL yang Anda berikan.

Tentu saja, pemrosesan tersebut tidak sesederhana bagaimana kita memeriksa input dengan if misalnya. Karena, kita tidak akan tahu persis bagaimana seseorang menuliskan kode program atau sintaks SQL misalnya. Ada yang menuliskan dalam beberapa baris untuk satu perintah, ada yang menggunakan banyak tab dan spasi, ada yang menuliskannya sedempet mungkin, dan lain sebagainya. Belum lagi kalau kita mengizinkan adanya bagian opsional dari suatu sintaks. Apa yang bisa kita lakukan adalah dengan memberikan aturan. Sebagai contoh, aturan-aturan fungsi dalam bahasa pemrograman tertentu. Atau, aturan sintaks SQL untuk mengambil data. Sebagai contoh, berikut ini adalah aturan sintaks SELECT dalam PostgreSQL:

```
SELECT [ ALL | DISTINCT [ ON (
expression [, ...] ) ] ]
```

```
* | expression [ AS output_
name ] [, ...]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY expression
[, ...] ]
[ HAVING condition [, ...] ]
[ { UNION | INTERSECT | EXCEPT
} [ ALL ] select ]
[ ORDER BY expression [ ASC |
DESC | USING operator ]
[, ...] ]
[ LIMIT { count | ALL } ]
[ OFFSET start ]
[ FOR UPDATE [ OF table_name
[, ...] ] ]
```

Setelah aturan kita definisikan, maka kemudian kita dapat mengatur sejumlah parser untuk menangani input dari user. Bicara soal parser itu sendiri, banyak sekali tool parser yang ada di dunia *open source*. Parser-parser generator tersebut dapat dimanfaatkan apabila Anda berniat untuk membangun aturan bahasa sendiri. Umumnya, apabila Anda ingin membangun bahasa pemrograman sendiri.

Di artikel ini, kita tidak akan membahas parser-parser tersebut. Apa yang akan kita bahas adalah pembuatan parser sederhana dengan memanfaatkan *regular expression*. Bahasa pemrograman yang akan dipergunakan adalah bahasa PHP. Dan, tentu saja, karena regular expression bekerja dengan

sangat lambat, maka kecepatan parsing akan jauh kalah dibandingkan kalau parsing dilakukan oleh tool yang sebenarnya.

Beberapa contoh yang akan kita bahas adalah pemeriksaan apakah suatu sintaks benar atau salah. Tentu saja, sintaks tersebut kita buat sesuai aturan yang kita inginkan. Setelah itu, kita juga bisa memanfaatkan regular expression untuk mengambil teks tertentu yang memenuhi sejumlah aturan (yang kita buat sendiri) untuk diproses lebih lanjut. Contoh yang paling nyata adalah ketika kita ingin membangun sendiri dokumentor dari source code kita.

Regular expression yang akan kita bahas mencakup posix extended RE dan Perl Compatible RE, sebagaimana yang didukung oleh PHP. Versi PHP yang digunakan adalah 4.x.

Posix Extended Regular expression

Posix extended RE adalah implementasi PHP untuk regular expression POSIX yang diatur dalam POSIX 1003.2. Umumnya, program-program di Linux yang kompatibel dengan standar POSIX juga menerapkan regular expression yang mengacu kepada standar POSIX tersebut. Contoh yang paling baik adalah grep dan tr.

Di PHP, umumnya Posix RE lebih mudah digunakan daripada PCRE. Sayangnya, Posix RE PHP tidaklah *binary-safe*. Namun, untuk pemrosesan teks yang kompleks sekalipun, Posix RE sudah jauh lebih dari cukup.

Umumnya, PHP yang terinstal pada sistem Anda telah dilengkapi dengan dukungan untuk regular expression ini, sehingga instalasi tambahan tidak diperlukan. Implementasi Posix RE di PHP dilakukan melalui fungsi-fungsi berikut ini:

- `ereg_replace` (untuk mengganti teks berdasarkan pattern RE).
- `ereg` (untuk mencari teks berdasarkan pattern RE).
- `eregi_replace` (sama seperti `ereg_replace`, namun bekerja secara case insensitive).
- `eregi` (sama seperti `ereg`, namun bekerja secara case insensitive).
- `split` (memecah teks menjadi array berdasarkan pattern RE).
- `spliti` (sama seperti `split`, namun bekerja secara case insensitive).
- `sql_regcase` (pembuatan RE untuk pencarian case insensitive).

Dengan fungsi-fungsi tersebut (yang sebenarnya sangat sedikit, hanya ada tiga sampai empat fungsi kalau tidak memperhatikan variannya), Anda sudah bisa membangun pemrosesan teks yang luar biasa menggunakan RE. Berikut ini adalah beberapa contoh sederhana.

ereg dan eregi

Andai kata Anda memiliki sejumlah teks, di mana Anda ingin mengetahui apakah terdapat teks tertentu di dalam teks tersebut yang memiliki aturan tertentu, bisa menggunakan fungsi ini. Secara umum, apabila terdapat kemungkinan bahwa teks dituliskan tanpa memperhatikan case (dan ini sangat umum terjadi), maka penggunaan `eregi()` akan lebih aman dan akurat.

Contoh sederhana adalah bagaimana `eregi()` digunakan untuk mengetahui apakah tanggal yang dimasukkan oleh user telah memenuhi aturan tanggal YYYY-MM-DD atau tidak. Berikut ini adalah kodenya:

source code:

```
<?
...
$pat = "([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})";
$input1 = "2005-06-19";
$input2 = "2005-6-19";
$input3 = "05-06-19";
```

```
if (ereg($pat, $input1))
    echo "$input1 sesuai dengan aturan $pat";
else
    echo "$input1 TIDAK sesuai dengan aturan $pat";
echo "\n";

if (ereg($pat, $input2))
    echo "$input2 sesuai dengan aturan $pat";
else
    echo "$input2 TIDAK sesuai dengan aturan $pat";
echo "\n";

if (ereg($pat, $input3))
    echo "$input3 sesuai dengan aturan $pat";
else
    echo "$input3 TIDAK sesuai dengan aturan $pat";
echo "\n";
?>
```

Penjelasan kode:

- Kita memiliki tiga variabel, masing-masing adalah `$input1`, `$input2` dan `$input3`. Ketiga variabel tersebut akan diuji dengan pattern `$pat`.
- Fungsi `ereg()` akan mengembalikan nilai true atau false sesuai dengan hasil pemeriksaan RE berdasarkan `$pat` kepada `$input1`, `$input2`, atau `$input3`.

Berikut ini adalah keluaran dari kode tersebut:

```
$ php a.php
2005-06-19 sesuai dengan aturan ([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})
2005-6-19 sesuai dengan aturan ([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})
05-06-19 TIDAK sesuai dengan aturan ([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})
```

Contoh tersebut tentunya bisa dikembangkan lebih lanjut menjadi contoh pemeriksaan sintaks sederhana. Contoh

berikut ini akan menguji apakah teks yang diberikan mengandung penulisan fungsi PHP yang valid dengan asumsi fungsi PHP yang valid adalah:

- diawali dengan kata `function`.
- diikuti oleh nama fungsi.
- diikuti oleh kurung buka.
- diikuti oleh variable list (opsional).
- diikuti oleh kurung tutup.
- diikuti oleh kurung kurawal buka.
- diikuti oleh isi fungsi (opsional).
- diikuti dan ditutup oleh kurung kurawal tutup.
- antar token yang diperiksa boleh dipisahkan dengan white space.

Namun, kita tidak akan memeriksa apakah sintaks dalam variable list dan isi fungsi adalah sintaks yang benar atau tidak.

source code:

```
<?
...
$pat = "^(function)([[:space:]]+)([a-zA-Z_]+([[:alnum:]]*)?)([[:space:]]+)(\(\)(.*)\)([[:space:]]+)]*)\{(.*)\}$";

$input1 = "function a()
{
    echo 'a';
}";

$input2 = "function _b($a, $b, $c)
{ print_r($a); print_r($b); print_r($c); }";

$input3 = "function 1a()
{ echo 'ini function yang salah karena nama fungsi diawali bilangan';}";

$input4 = "function z()";

if (eregi($pat, $input1))
    echo "Syntax OK";
else
    echo "Syntax Error";
echo "\n";

if (eregi($pat, $input2))
    echo "Syntax OK";
else
```

```

echo "Syntax Error";
echo "\n";

if (ereg($pat, $input3))
    echo "Syntax OK";
else
    echo "Syntax Error";
echo "\n";

if (ereg($pat, $input4))
    echo "Syntax OK";
else
    echo "Syntax Error";
echo "\n";

?>

```

penjelasan source code:

- Kita memiliki satu *pattern* untuk pengecekan sintaks penulisan fungsi di PHP yang disimpan dalam variabel \$pat. Sekali lagi, ini adalah pengecekan sederhana dan tidak mengecek kebenaran sintaks *variable list* dan *function body*.
- Kita memiliki empat variabel input yang masing-masing berisikan string fungsi yang ingin diuji.
- \$input1 dan \$input2 adalah contoh yang valid
- \$input3 dan \$input4 adalah contoh yang tidak valid karena pada \$input3, nama fungsi diawali dengan bilangan. Sementara, \$input4 tidak valid karena deklarasi fungsi belum memasukkan penanda isi fungsi.
- Khusus untuk \$pat, kita memecah token-token yang ingin diuji ke dalam group-group, yang ditandai dengan kurang buka dan kurang tutup. Adanya `[[:space:]]` diantara beberapa token menandai bahwa white space diijinkan diantara token-token tersebut.
- repetition operator `*` berarti nol atau lebih, repetition operator `+` berarti satu atau lebih.
- Karakter `.` (titik) mewakili setiap karakter.
- Beberapa karakter seperti `(` dan `{` perlu diescape menjadi `\(` dan `\{` karena merupakan karakter spesial yang memiliki arti tertentu di dalam RE.

Berikut ini adalah keluaran dari kode tersebut:

```

$ php b.php
Syntax OK
Syntax OK
Syntax Error
Syntax Error

```

Fungsi `ereg()` dan `eregi()` merupakan fungsi yang sangat berguna, seperti dalam kedua contoh tersebut. Anda bisa membangun rutin untuk memeriksa pattern yang definisikan sebagai aturan Anda sendiri. Untuk membantu, POSIX RE telah menyediakan beberapa kelas karakter. Berikut ini adalah beberapa di antaranya:

- `[alnum:]`, semua huruf dan bilangan.
- `[alpha:]`, semua huruf.
- `[blank:]`, semua white space horizontal.
- `[digit:]`, semua bilangan.
- `[print:]`, semua karakter yang bisa dicetak, termasuk spasi.
- `[space:]`, semua white space vertikal dan horizontal.

Kelas-kelas tersebut bisa digunakan di program `grep`, `tr` dan program lain yang memanfaatkan Posix RE.

ereg_replace dan eregi_replace

Fungsi `ereg()` dan `eregi()` berguna untuk melakukan *matching*. Namun, ada kalanya, *matching* saja tidak cukup. Contoh yang umum terjadi misalnya adalah ketika Anda ingin mengganti teks tertentu dalam suatu teks. Apabila menggunakan `ereg()` atau `eregi()`, maka hanya dilakukan pencocokan terhadap pattern tertentu. Sementara, apabila Anda ingin langsung melakukan pengubahan, maka Anda harus menggunakan fungsi `ereg_replace()` ataupun `eregi_replace()`.

Contoh sederhana berikut ini adalah bagaimana kita mengganti semua `a` kecil menjadi `A` besar. Berikut ini adalah source code-nya:

```

<?
$pat_src = "a";
$pat_dst = "A";
$src = "Hari ini saya ke pasar baru untuk membeli buku bekas";
$dst = eregi_replace($pat_src, $pat_dst, $src);
echo "src:\n$src\n";

```

```

echo "dst:\n$dst\n";

?>

```

Penjelasan kode:

Dengan menggunakan `ereg_replace()`, kita mengganti semua `a` ke `A`. Fungsi `ereg_replace()` akan menerima tiga parameter, yaitu *pattern*, *replacement* dan *string asal*, serta akan mengembalikan string hasil penggantian.

Satu hal yang harus diperhatikan dalam penggunaan `ereg_replace()` ataupun `eregi_replace()` adalah masalah penggantian teks dengan bilangan. Sebagai contoh, Anda ingin mengganti tulisan satu menjadi angka 1. Apabila Anda memberikan 1 sebagai *replacement*, maka hasil penggantian mungkin tidak akan berjalan sesuai yang diinginkan. Agar hasil penggantian benar, Anda harus menggantinya dengan karakter `'1'`.

split dan spliti

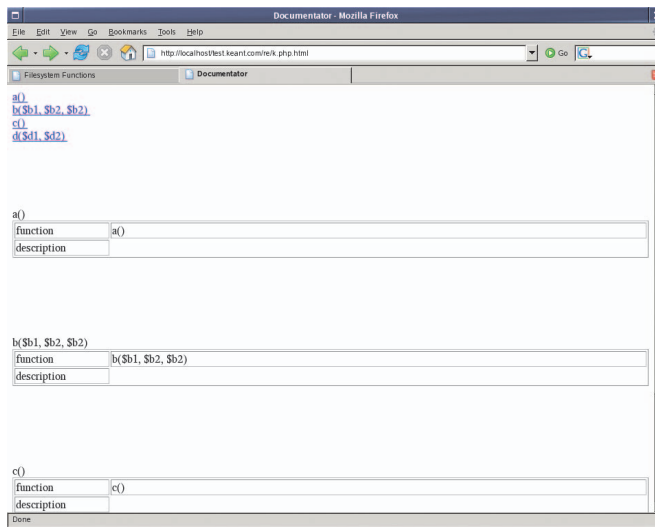
Terkadang, dalam beberapa kasus pemrograman, ada kalanya kita perlu memisahkan elemen teks berdasarkan pemisah tertentu ke dalam array. Tentunya, memproses array akan jauh lebih mudah daripada kita memproses teks begitu saja. Contoh kasus yang umum adalah ketika kita ingin memproses entri di `/etc/passwd`.

Berikut ini adalah contoh untuk mendapatkan username dan home directory semua user di `/etc/passwd`:

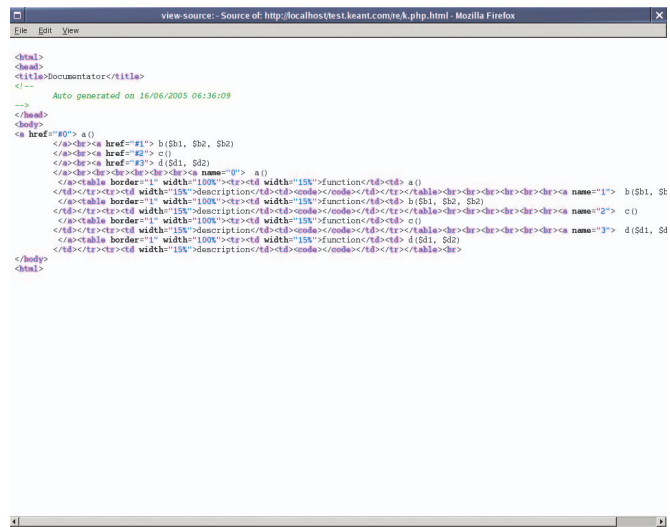
```

<?
$f_passwd = fopen("/etc/passwd", "r");
while (!feof($f_passwd))
{
    $buff = fgets($f_passwd);
    if (strlen($buff) > 1)
    {
        $user_arr = split(":", $buff);
        echo "user: {$user_arr[0]}, homedir: {$user_arr[5]}\n";
    }
}
fclose($f_passwd);
?>

```



HTML hasil dokumentator.php.



Source code HTML hasil dokumentator.php.

Berikut ini adalah keluaran program tersebut di komputer penulis:

```
$ php d.php
user: root, homedir: /root
user: bin, homedir: /bin
user: daemon, homedir: /sbin
user: lp, homedir: /var/spool/lpd
user: mail, homedir: /var/spool/clientmqueue
user: news, homedir: /etc/news
user: uucp, homedir: /etc/uucp
user: games, homedir: /var/games
user: man, homedir: /var/cache/man
user: at, homedir: /var/spool/atjobs
user: wwwrun, homedir: /var/lib/wwwrun
user: ftp, homedir: /srv/ftp
user: postfix, homedir: /var/spool/postfix
user: sshd, homedir: /var/lib/ssh
user: messagebus, homedir: /var/run/dbus
user: haldaemon, homedir: /var/run/hal
user: nobody, homedir: /var/lib/nobody
user: nop, homedir: /home/nop
user: postgres, homedir: /var/lib/pgsql
```

Karena split() menerima pattern RE sebagai pemisah, maka pemisah tidak harus

selalu berupa karakter pasti. Contoh yang umum terjadi adalah ketika pemisahan ingin dilakukan berdasarkan white space. Dalam kasus tersebut, kita tidak bisa hanya menentukan berdasarkan spasi, atau tab, ataupun kombinasi keduanya. Memecah entri pada /etc/fstab adalah contoh yang baik. Berikut ini adalah contohnya:

```
<?
$fs_fstab = fopen("/etc/fstab", "r");
while (!feof($fs_fstab))
{
    $buff = fgets($fs_fstab);
    if (strlen($buff) > 1)
    {
        $fs_arr = split("[[:space:]]+", $buff);
        echo "device: {$fs_arr[0]}, mount point: {$fs_arr[1]}\n";
    }
    fclose($fs_fstab);
?>
```

Penjelasan kode:

Karena kita ingin memisahkan berdasarkan white space, maka kita perlu menggunakan kelas karakter [[:space:]]. Adalah penting untuk menambahkan + agar white space dikelompokkan (bukannya satu per satu spasi misalnya).

Dengan beberapa fungsi saja yang datang dengan Posix RE, kita bisa membangun

parser sederhana ataupun tool-tool lain yang bisa membantu kita bekerja lebih cepat.

PCRE

Pattern PCRE umumnya cenderung lebih susah untuk dibaca. Namun, PCRE sendiri pada PHP seringkali dikatakan lebih cepat daripada implementasi Posix RE. Dan, implementasi PCRE PHP telah binary safe. Dalam beberapa kasus yang lebih berat, umumnya PCRE lebih disukai daripada Posix RE.

Implementasi PCRE di PHP datang dengan fungsi-fungsi berikut:

- preg_grep, mengembalikan array yang sesuai dengan pattern.
- preg_match_all, memeriksa seluruh teks
- preg_match, memeriksa teks.
- preg_quote, melakukan quoting pada karakter RE.
- preg_replace_callback, melakukan pencarian RE dan melakukan penggantian dengan fungsi callback.
- preg_replace, mencari dan mengganti teks.
- preg_split, memecah teks dan menyimpannya ke dalam array.

Berbeda dengan Posix RE, PCRE mengenal Pattern Modifier, modifier untuk pattern RE. Di sini, kita bisa menentukan apakah pencarian akan dilakukan case sensitive dan lain sebagainya.

Satu catatan, walaupun mengusung nama PCRE, implementasi PCRE di PHP memiliki beberapa perbedaan dengan Perl. Anda bisa membaca lebih lanjut pada Do-

kumentasi PHP pada bagian Regular Expression Functions (Perl-Compatible), sub-bagian Pattern syntax.

preg_match dan preg_match_all

Fungsi ini berguna untuk melakukan matching pattern RE pada teks. Kita akan membuat ulang contoh yang kita bahas pada pembahasan fungsi ereg() dan eregi() sebelumnya. Yang pertama adalah bagaimana memeriksa apakah input tanggal telah memenuhi aturan YYYY-MM-DD atau tidak.

Berikut ini adalah source code-nya:

```
<?

$pat = "/([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})/i";
$input1 = "2005-06-19";
$input2 = "2005-6-19";
$input3 = "05-06-19";

if (preg_match($pat, $input1))
    echo "$input1 sesuai dengan aturan $pat";
else
    echo "$input1 TIDAK sesuai dengan aturan $pat";
echo "\n";

if (preg_match($pat, $input2))
    echo "$input2 sesuai dengan aturan $pat";
else
    echo "$input2 TIDAK sesuai dengan aturan $pat";
echo "\n";

if (preg_match($pat, $input3))
    echo "$input3 sesuai dengan aturan $pat";
else
    echo "$input3 TIDAK sesuai dengan aturan $pat";
echo "\n";

?>
```

Penjelasan kode:

- Bisa dilihat bahwa apa yang berubah pada contoh versi Posix RE adalah pattern dan penggunaan fungsi preg_match()

- Pattern modifier yang kita gunakan adalah i, yang artinya pencarian akan dilakukan case insensitive (sama seperti penggunaan fungsi eregi()).

Selanjutnya, kita akan membuat ulang kode untuk memeriksa keabsahan definisi fungsi PHP:

```
<?

$pat = "/^(function)(\s*?)
([a-zA-Z_]+\w*)(\s*?)
(\)(.*)?(\s*?)
(\{(.*)\})$/mis";

$input1 = "function a()
{
    echo 'a';
}";

$input2 = "function _b($a, $b,
$c)      { print_r($a); print_r($b); print_r($c); }";

$input3 = "function -a()
{ echo 'ini function yang salah karena nama fungsi diawali bilangan';}";

$input4 = "function z()";

if (preg_match($pat, $input1))
    echo "Syntax OK";
else
    echo "Syntax Error";
echo "\n";

if (preg_match($pat, $input2))
    echo "Syntax OK";
else
    echo "Syntax Error";
echo "\n";

if (preg_match($pat, $input3))
    echo "Syntax OK";
else
    echo "Syntax Error";
echo "\n";

if (preg_match($pat, $input4))
    echo "Syntax OK";
else
    echo "Syntax Error";
```

```
echo "\n";
```

```
?>
```

Penjelasan kode:

- Kita menggunakan pattern modifier mis, yang artinya adalah:
 - ➔ m untuk multiline.
 - ➔ i untuk case insensitive.
 - ➔ s untuk membuat metakarakter . (titik) memasukkan karakter *newline*.
- Beberapa kelas karakter pada Posix RE umumnya ditulis lebih singkat pada versi PCRE

preg_replace

Fungsinya sama dengan ereg_replace, namun menurut penulis, fungsi yang satu ini lebih enak untuk digunakan. Kita akan melihat contoh bagaimana mengganti tulisan [date] di dalam string menjadi tanggal aktif.

Berikut ini adalah source code-nya:

```
<?

$src = "Hari ini tanggal [date].";

$dst = preg_replace("/([date])/mis", date("d-m-Y"), $src);
echo $dst . "\n";

?>
```

preg_split

Fungsi ini berguna untuk memecah string ke dalam array berdasarkan pattern tertentu. Fungsinya sama dengan split() atau spliti() milik Posix RE. Berikut ini adalah penulisan ulang contoh untuk membaca /etc/fstab:

```
<?

$f_fstab = fopen("/etc/fstab", "r");
while (!feof($f_fstab))
{
    $buff = fgets($f_fstab);
    if (strlen($buff) > 1)
    {
        $fs_arr = preg_split("/\s+/", $buff);
        echo "device: {$fs_arr[0]}, mount point: {$fs_arr[1]}\n";
    }
}
```



```

}
fclose($f_fstab);
?>

```

Contoh kasus: Dokumentator

Setelah melihat contoh-contoh penggunaan RE, kita akan membahas satu contoh kasus sederhana, yaitu program dokumentator yang akan mengambil source PHP dan mengekstrak dokumentasi fungsi dan memformatnya menjadi satu file HTML yang berisi dokumentasi fungsi. Untuk itu, kita akan membuat aturan baru dokumentasi kita. Berikut ini adalah aturan dokumentasi:

- Diawali dengan `/**` dan diakhiri dengan `*/`
- Tersedia deskripsi fungsi yang dimungkinkan dengan pemberian `@desc:` `<deskripsi fungsi>`;

Dengan demikian, berikut ini adalah contoh dokumentasi di dalam satu fungsi:

```

function a()
{
    /**
     * @desc:
     * - deskripsi pertama fungsi
     * - deskripsi kedua fungsi
     * ;
     */
    //isi fungsi
}

```

Berikut ini adalah source code dokumentator kita:

```

<?
$index_header = "
<html>
<head>
<title>Documentator</title>
<!--
    Auto generated on " . date("d/
m/Y h:i:s").
"
-->
</head>
<body>
";

$index_footer = "
</body>
<html>

```

```

";

echo "dokumentator.php, (c) Nop
19 June 2005\n";
$f = $_SERVER['argv'][1];
if ( !$f || !file_exists($f) )
die ("usage: dokumentator.php
<script>\n");

$f_out = $f . ".html";
$str = fread(fopen($f, "r"),
filesize($f));
$pat = '/(function)(.*?)(\{)(.*?
)(\})/mis';
preg_match_all($pat,$str,
$matches);

for ($i = 0; $i<count
($matches[0]); $i++)
{
    //get comments from file,
    parse comments and function
    body
    $pat_comm = '/(\\\/\*.*)(.*?)(\\
*\\/)(.*)/mis';
    preg_match_all($pat_comm,
    $matches[4][$i],$matches_
    comm);
    (count($matches_comm[0])<1)
    ? $f_body = $matches[4][$i] :
    $f_body = $matches_comm[4][0];

    //already got comments from
    file, parse specific comment
    //get desc
    $pat_desc = '/(.*?)(@desc)
(:)(\s*?)(.*?)(;)/mis';
    preg_match_all($pat_desc,
    $matches_comm[2][0],$matches_
    desc);

    $f_name = $matches[2][$i];
    $f_desc = nl2br(trim($matches_
    desc[5][0]));

    //link creation
    $link_info .= "<a href=\"#$i\
\">$f_name</a><br>";

    //detail information per
    function
    $detail_info .=
    "<br><br><br><br>";

```

```

$detail_info .= "<a name=\"
\"#$i\"> $f_name </a>";
$detail_info .= "<table
border=\"1\" width=\"100%\">";
$detail_info .= "<tr><td
width=\"15%\">function</
td><td>\" . $f_name .
\"</td></tr>";
$detail_info .= "<tr><td
width=\"15%\">description
</td><td><code>\" . $f_desc .
\"</code></td></tr>";
$detail_info .= "</table><br>";

}
$index_body = $index_header .
$link_info . $detail_info .
$index_footer;
fwrite(fopen($f_out,"w"),
$index_body);
echo "DONE\n";
?>

```

Penjelasan kode:

- Prinsip kerjanya, pertama-tama kita akan memeriksa terlebih dahulu apakah argumen pertama tersedia, dan apabila tersedia, merupakan file yang valid atau tidak. Setelah itu, apabila semuanya valid, maka proses pun dilanjutkan.
- Setelah itu, kita akan mengambil semua dokumentasi dari fungsi tersebut.
- Setelah itu, dari semua hasil yang berhasil diambil, kita akan mengambil deskripsi dari komentar tersebut, membuat link dan detil informasi fungsi.
- Setelah itu, kita menggabungkan semua yang didapat dan menuliskannya ke file HTML.

Program ini tentunya masih bisa disempurnakan, misal dengan menambahkan informasi lain seperti return type, contoh penggunaan fungsi, dan lain sebagainya. Tentunya, tampilan HTML hasil pembuatan juga bisa dimodifikasi sesuai keinginan.

Sampai di sini dulu pembahasan kita tentang parsing sederhana menggunakan RE. Untuk tugas parsing yang kompleks, RE mungkin bukan solusi yang tepat. Tapi, untuk parsing sederhana, RE jauh lebih dari cukup. Selamat mencoba! 🙌

Noprianto (noprianto@infolinux.co.id)