Ek B

Düzenli İfadeler

Düzenli ifadeler (regular expressions), bir metin içinde belli bir düzene uyan bir kelime dizisini bulmak için kullanılır. Teknik olarak bu işlemin sonucundan bazen şu şekilde bahsedilir: "xyz düzenli ifadesi abc ifadesine uydu (matched)". İfade, aslında bir kelime dizisini temsil eden genelci bir beyandır; Aslında komut satırından bir düzenli ifadeyi sürekli kullanıyoruz. '`*`' işareti. Bu işareti bir listeleme komutu ile beraber kullandığımızda, '`*`' işareti "ne olursa olsun" düzenli ifadesi olduğu için, bu tarife uyan tüm dosyaları ve dizinleri geri almış oluruz.

Fakat düzenli ifade '`*`' işaretinden ibaret değildir. Çok daha güçlü ve sayesinde değişik ve çetrefil metinler çekip çıkarabileceğimiz ifadeler vardır. Ama ondan önce, üzerinde düzenli ifade işleyeceğimiz girdi dosyalarını nasıl okuyacağımızı görmemiz gerekiyor, yâni Perl'ün dosya erişim özelliklerini.

B.1 Perl ile Metin İşleme

Perl'ün metin işleme yeteneklerini bir örnek üzerinde görelim: Perl ile dosya açmak için `open` komutu kullanılır.

```
open IN, ‘‘fileName.txt’’;
```

Bu komutla `fileName.txt` okunmak için açılmış olur. Bir dosyayı satır satır, ya da tamamen okuyabiliriz. Çoğunlukla (meselâ kodda) yapılan metin değişiklikleri için dosyanın tamamını aynı anda okumak gerekir. Bunu yapmak için, `undef \$/;` işareti kullanmamız gerekecektir. Bu tanımdan sonra tüm dosya içeriğini bir değişkene atamak için, şu kullanılır.

```
\$fileContent = <IN>;
```

Bu yapıldıktan sonra `\$fileContent` değişkeni, dosyadaki tüm içeriği içinde barındırmaya başlar. Kodlamanızı daha kısa tutmak istiyorsanız, `\$fileContent` yerine özel bir değişken olan `\$_` değişkenini de kullanabilirsiniz. Gizli değişkeni kullanmanın iyi taraflarından biri, takip eden tüm düzenli ifade işlemlerinin (eğer onlar da değişken belirtmemişse) otomatik olarak gizli değişken üzerinde yapılabilmesidir. Çok basit örnek bir düzenli ifade kullanmak gerekirse

```
open IN, ‘‘fileName.txt’’;
\$_ = <IN>;
s/ahmet/mehmet/sg ;
print;
close IN;
```

Bu örnekte, “`ahmet`” kelimesini “`mehmet`” ile değiştirdik, ve ekrana bastık. `print` komutuna hiç parametre vermezseniz, aynı şekilde gizli değişken olan `\$_` kullanıldığını farz edecektir.

İlk düzenli ifademiz, hem bulan, hem değiştiren bir ifade. Bu ifadenin `s/¬ string1/string2/sg;` tabirinde olan `string1`, aranan kısımdır, `string2` ise, aranan ifade bulununca, yerine geçecek kısımdır. Değiştirmek yerine sadece bulmak istiyorsanız, `string1` ifadesini sadece `if` ile kullanabilirsiniz; `if (/¬ string1/) {.. .}` gibi.

B.1.1 Çıktı Dosyası

Metin değişimi yapmak için genellikle değiştirilen dosya ikinci bir dizin altında aynı isimde yazılır.

```
open IN, ‘‘fileName.txt’’;
open OUT, ‘‘/tmp/fileName.txt’’;
\$_ = <IN>;
s/ahmet/mehmet/sg ;
print OUT;
close IN;
close OUT;
```

Bu kod parçasındaki `OUT`, değişmiş içeriğimizin gittiği dosya olacaktır. Dosyayının aynı isimle açıldığı dizinin değişik bir yerde olduğuna dikkat edelim.

B.1.2 Birçok Giriş Dosyası

Eğer bir dizin altındaki birçok dosyayı işlemek istiyorsak, dosya listesi almak için `<>` işaretlerini kullanabiliriz. Bu işaretlerin arasında, çoğu işletim sisteminden alışık olduğumuz ‘`*`’ kullanımı mümkündür. `<>` kullanımında geriye bir Perl listesi gelecektir, bu listeyi `foreach` Perl komutu ile gezebiliriz. Meselâ

```
foreach \$file(<*.java>) {
  # ..
  # \$file ile işlemler yap
```

```
  # ..
}
```

Eğer biraz önceki değiştirme mantığını üstte gösterilen tüm dosyalara uygulamak istersek,

```
foreach \$file(<*.java>) {
  open IN, ‘‘\$file’’;
  open OUT, ‘‘/tmp/\$file’’;
  \$_ = <IN>;
  s/ahmet/mehmet/sg ;
  print OUT;
  close IN;
  close OUT;
}
```

Dosyaların okunacağı dizini değiştirmek için ise, Perl `chdir` komutu kullanılabilir; `chdir(‘‘/vs/vs’’)` gibi.

B.2 İfadeler

Artık daha zor düzenli ifadeleri gösterebiliriz. Bu yeni daha çetrefil ifadeleri `//` arasına yerleştirirsek, bu yeni ifadeleri kullanmiş oluruz. Böylece o ifadenin temsil ettiği metne göre, o anda okunmakta olan dosya üzerinde bu verilen ifade *uydurulmaya* çalışılacaktır. Uydurulan (bulunan) ifade ise, otomatik olarak değiştirme bölümünü devreye sokacaktır.

Düzenli ifadeler içinde her çeşit metin dizisi için, bir komut vardır. Meselâ eğer `22.33.22`, ya dâ, `33.22.44` yâni, genel olarak, “iki sayı, sonra bir nokta, iki sayı daha, sonra nokta ve son olarak iki sayı” içeren bir *düzeni* bulmak istersek, o zaman tek vermemiz gereken düzenli ifade şu olacaktır.

```
/\d{2}\.\d{2}\.\d{2}\./
```

Burada `\d` komutu, tek haneli bir sayıyı temsil etmektedir. Tek karakteri temsil eden tüm özel komutların listesi Tablo B.1 üzerinde bulunabilir.

Tablo B.1 komutları, elle gömeceğimiz sabit metin ile aynı ifadenin parçası olarak kullanılabilir. Ayrıca çoğu zaman, karakter komutlarını “bazı özel tekrar etme kuralları” ile beraber kullanmak gerekiyor. Bu özel tekrar kuralları, genellikle, bir düzenli ifade komutunun hemen yanına eklenir, ve yanına eklendiği tabirin temsil gücünü daha da arttırır. Tablo B.2 üzerinde tekrar kurallarını görüyoruz.

Örnekler

`\$_ = ‘‘abbbccdaabccdde’’` üzerinde bazı düzenli ifade örnekleri görelim[1]:

[1] Can Uğur Ayfer, `http://www.mycompany.com/localhost/mycompany/yazi.jsp@dosya=a_regular_expression.xml.html`

Tablo B.1: Düzenli İfade Komutları

Komut	**Târif**
`\d`	Tek haneli bir sayı
`\D`	Tek haneli haricinde her şey
`\w`	Bir alfabe harfi (a ile z, A ile Z arası, ve _ işareti)
`\w`	Alfabe haricinde herhangi bir karakter
`\s`	Beyaz boşluk (SPACE, TAB) karakteri
`\S`	Beyaz boşluk haricinde herhangi bir karakter
`\`	META (ESC) karakteri
`^`	Satır başlangıç karakteri
`.`	Herhangi bir karakter
`\$`	Satır sonu karakteri

Tablo B.2: Tekrar Komutları

Komut	**Târif**
`*`	0 ya da daha fazla
`+`	1 ya da daha fazla
`?`	1 ya da hiç
`{n}`	n kere
`{n,}`	en az ne kere
`{n,m}`	en az ne kere, ama m'den fazla değil
`str1 | str2`	Ya ifade1 ya da ifade2 bulunacak

- `/Abc/` Uymaz! `\$d` içinde hiç "A" yoktur yâni "Abc" bulunamaz.
- `/Abc/i` Uyar! Sondaki "i" ignore case, yani büyük-küçük harf ayırımı yapılmasın anlamında olduğu için "Abc" ile "abc" eşleşir.
- `/abc/` Uyar!
- `/^abc/` Uymaz! Çünkü "^" meta-karakteri dizinin başında anlamındadır ve "abc" alt dizisi `\$d`'nin basinda değildir.
- `/abc>/` Uymaz! Çünkü ">" meta-karakteri dizinin sonunda anlamındadır ve "abc" alt dizisi `\$d`'nin sonunda değildir.
- `/De>/i` Uyar! Dizimizin sonunda "de" var ve büyük-küçük harf ayırımı yapılmayacak!
- `/^ab*c/` Uyar, çünkü dizinin basinda "a" ve ardından "sıfır veya daha fazla b" ve ardından "c" var! Bu düzenli ifadede olan "*" karakteri hemen solundaki karakter için "sıfır veya daha fazla kez tekrar eden" anlamında bir meta karakterdir.
- `/aH*bc/` Uyar! Çünkü dizide "a" ve ardından sıfır veya daha fazla "H" ve ardinda "bc" var!
- `/aH+bc/` Uymaz! Çünkü dizide "a" ve ardından en az bir tane "H" ve onun ardında "bc" seklinde bir düzen yok! Bu Duzenli ifadede olan "+" karakteri hemen solundaki karakter için "bir veya daha fazla kez tekrar eden" anlamında bir meta-karakterdir.
- `/a*bc/` Uymaz! Çünkü dizinin içinde hiç "a" ve ardından gelen "*" yok! Bu örnekte "*" bir meta-karakter olarak değil; basit anlamıyla bir asterisk karakteri olarak kullanıldığı için önündeki "" ile işaretlenmiştir.
- `/a+bc/` Uymaz! Çünkü dizinin içinde hiç "a" ve ardindan gelen "+" yok! Bu örnekte "+" bir meta-karakter olarak değil; basit anlamıyla bir artı işareti olarak kullanıldığı için önündeki "" ile işaretlenmiştir.
- `/a c/` Uymaz! Çünkü dizinin içinde hiç "a" ve ardindan gelen "" yok!
- `/a.*b/` Uyar! Çünkü dizide "a ve aralarında birşeyler ve sonra b" var! "." (nokta) meta-karakteri herhangi bir karaktere uyar; ardından gelen "*" ile birlikte (yâni "`.*`") birşeyler olarak okunabilir.
- `/d.*a/` Uyar! Çünkü "birşey" anlamındaki noktanın ardından gelen "*" meta-karakteri sıfır veya daha fazla herhangi birşey anlamındadır.
- `/d.+a/` Uyar! Çünkü "birşey" anlamındaki noktanın ardından gelen meta-karakter bir "`+`"; Yâni bir veya daha fazla "herhangi birşey"
- `/da?/` Uyar! Çünkü dizide "d" ve ardından bir veya sıfır tane "a" gelen alt dizi var.

B.3 Gruplama ve Bulunanı Kullanmak

Çoğu değiştirme işlemi için, bulmak için kullandığımız düzenli ifadenin bulduğu metnin "bir bölümünü" yeni değiştirmenin bir parçası olarak kullanmamız gerekebilir. Meselâ, alttaki gibi bir kod parçası üzerinde

```
import com.sirket.paket.vs;

if (bilmemne)
   soyle boyle;

soyle boyle;

if (filan)
   soyle boyle oyle;
```

yapacağımız değişiklik şöyle olabilir; "sadece `if` komutunu içeren kod satırları (`;` işareti ile biten kod satırları, dosya satırları değil) içinde bütün "şöyle böyle" ibarelerini, "böyle şöyle" yapalım". Bu gibi durumlarda, bulunan kelimenin kendisi yeni kelimenin bir parçasıdır, bu yüzden bir şekilde onlara erişebilmemiz gerekir. Bunun için Perl'ün `()` gruplama işaretlerini kullanabiliriz. Perl'de her düzenli ifade komutu, parantez içine alınabilir. Alındığı zaman, ve bu ifade metin parçası ile uyuştuğu zaman (matching), parantez içindeki uyan kısım, değiştirme yapan (ikinci `//` arasındaki komutlar) kısımda, gruplamanın kullanılış sırasına göre, `\$1`, `\$2`, `\$3`, .. ile erişilebilir. Meselâ örneğimizde üç tane `()` kullanımı var, birinci uyan kısma erişmek için `\$1` kullanacağız.

```
s/\;(.*?)if (.*?) soyle boyle(.*?)\;/\;\$1if \$2 boyle soyle\$3\;/sg;
```