

# Regex Kütüphanesi Başvuru Kılavuzu

Çeviren:

**Yalçın Kolukısa**

<yalcink01 (at) yahoo.com>

27 Şubat 2007

## Özet

Bu belge `regex-0.12` paketiyle (1993 yılında bağımsız geliştirilmesi durdurulmuş ve GNU C kütüphanesine dahil edilmiştir) dağıtılmış olan Regex Kütüphanesi Başvuru Kılavuzunun çevirisidir. GNU C Kılavuzunda Regex oluşumları ile ilgili bilgiler bu belgedeki kadar ayrıntılı değildir. Bu nedenle, bu belge GNU C Kütüphanesi Başvuru Kılavuzunun tamamlayıcısı olarak ele alınmalıdır.

## İçindekiler

<b>1. Genel Bakış</b>	4
<b>2. Düzenli İfade Sözdizimi</b>	4
2.1. Sözdizimi Bitleri	4
2.2. Öntanımlı Sözdizimleri	6
2.3. Elemanların ve Karakterlerin Alfabetik Sıralaması	7
2.4. Tersbölü Karakteri	8
<b>3. Ortak İşleçler</b>	8
3.1. Kendisiyle Eşleşme İşleci	9
3.2. Herhangi Bir Karakterle Eşleşme İşleci	9
3.3. Birleştirme İşleci	9
3.4. Yineleme İşleçleri	9
3.4.1. Sıfır veya daha fazlası ile eşleşme işleci	9
3.4.2. Bir veya daha fazlası ile eşleştirme işleci	10
3.4.3. Sıfır veya bir kere eşleşme işleci	10
3.4.4. Sınırlı sayıda yineleme işleçleri	10
3.5. VEYA İşleci	11
3.6. Liste İşleçleri	11
3.6.1. Karakter Sınıfı İşleçleri	12
3.6.2. Aralık İşleci	13
3.7. Gruplama İşleçleri	13
3.8. Grup Adresleme İşleci	14
3.9. Demirleme İşleçleri	14
3.9.1. Satır başı ile eşleşme işleci	14
3.9.2. Satır sonu ile eşleşme işleci	15
<b>4. GNU İşleçleri</b>	15

4.1. Sözcük İşleçleri	15
4.1.1. Emacs-dışı Sözdizimi Tabloları	15
4.1.2. Sözcük sınırlarıyla eşleşme işleci (b)	16
4.1.3. Sözcük-içi eşleşme işleci (B)	16
4.1.4. Sözcük başlangıcıyla eşleşme işleci (<)	16
4.1.5. Sözcük sonuyla eşleşme işleci (>)	16
4.1.6. Sözcük bileşenleriyle eşleşme işleci (w)	16
4.1.7. Sözcük bileşeni olmayanlarla eşleşme işleci (W)	16
4.2. Tampon İşleçleri	16
4.2.1. Tampon başlangıcıyla eşleşme işleci (^)	16
4.2.2. Tampon sonuyla eşleşme işleci (\$)	16
<b>5. GNU Emacs İşleçleri</b>	16
5.1. Sözdizimsel Sınıf İşleçleri	16
5.1.1. Emacs Sözdizimi Tabloları	16
5.1.2. Sözdizimsel Sınıfları Eşleştirme İşleçleri	17
5.1.3. Sözdizimsel Olmayan Sınıfları Eşleştirme İşleçleri	17
<b>6. Ne Eşleştirilir?</b>	17
<b>7. Regex ile Yazılım Geliştirme</b>	17
7.1. GNU Regex İşlevleri	17
7.1.1. GNU Şablon Tamponları	17
7.1.2. GNU Düzenli İfadelerinin Derlenmesi	19
7.1.3. GNU Eşleştirme İşlevi	20
7.1.4. GNU Arama İşlevi	20
7.1.5. Veriyi Bölerek Arama ve Eşleştirme	21
7.1.6. Hızlı İşlemlerle Arama	22
7.1.7. GNU Çeviri Tabloları	22
7.1.8. Yazmaçların Kullanımı	23
7.1.9. GNU Şablon Tamponlarının Serbest Bırakılması	25
7.1.10. Bir Arama ve Değişirme Örneği	25
7.2. POSIX Regex İşlevleri	26
7.2.1. POSIX Şablon Tamponları	27
7.2.2. POSIX Düzenli İfadelerinin Derlenmesi	27
7.2.3. POSIX Eşleştirmesi	28
7.2.4. Hataların Bildirilmesi	29
7.2.5. Bayt Konumlarının kullanımı	29
7.2.6. POSIX Şablon Tamponlarının Serbest Bırakılması	30
7.3. BSD Regex İşlevleri	30
7.3.1. BSD Düzenli İfadelerinin Derlenmesi	30
7.3.2. BSD Araması	30
<b>A. Bu Kılavuzun Kopyalanması</b>	31
<b>Dizin</b>	37

## Legal Notice

Copyright © 1992–2005 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, with the Front-Cover texts being "A GNU Manual," and with the Back-Cover Texts as in (a) below. A copy of the license is included in the section entitled "GNU Free Documentation License."

(a) The FSF's Back-Cover Text is: "You have freedom to copy and modify this GNU Manual, like GNU software. Copies published by the Free Software Foundation raise funds for GNU development."

## Yasal Uyarı

Telif hakkı © 1992–2005 Free Software Foundation, Inc.

Bu kılavuzun harfi harfine kopyalanmasına ve dağıtılmasına telif hakkı uyarısının ve bu izin uyarısının tüm kopyalarında bulunması şartıyla izin verilmiştir.

Bu belgeyi; Free Software Foundation tarafından yayınlanmış olan GNU Özgür Belgelendirme Lisansının 1.2 veya daha sonraki bir sürümüne sadık kalmak koşulu ile kopyalayabilir, dağıtabilir veya düzenleyebilirsiniz: değişmez bölümler yoktur, ön-kapak yazısı olarak "A GNU Manual" ile aşağıdaki (a) şıkkındaki arka-kapak yazısı bulunmalıdır. Bu Lisansın bir kopyasını [GNU Free Documentation License](#) (sayfa: 31) başlıklı bölümde bulabilirsiniz.

(a) FSF'nin Arka-Kapak Metni: "You have freedom to copy and modify this GNU Manual, like GNU software. Copies published by the Free Software Foundation raise funds for GNU development."

## Feragatname

Belge içeriğindeki bilgileri uygulama sorumluluğu uygulayana aittir.

BU KİTAP ÜCRETSİZ OLARAK RUHSATLANDIĞI İÇİN, İÇERDİĞİ BİLGİLER İÇİN İLGİLİ KANUNLARINI İZİN VERDİĞİ ÖLÇÜDE HERHANGİ BİR GARANTİ VERİLMEMEKTEDİR. AKSİ YAZILI OLARAK BELİRTİLMEDİĞİ MÜDDETÇE TELİF HAKKI SAHİPLERİ VE/VEYA BAŞKA ŞAHISLAR KİTABI "OLDUĞU GİBİ", AŞIKAR VEYA ZIMNEN, SATILABİLİRLİĞİ VEYA HERHANGİ BİR AMACA UYGUNLUĞU DA DAHİL OLMAK ÜZERE HİÇBİR GARANTİ VERMEKSİZİN DAĞITMAKTADIRLAR. BİLGİNİN KALİTESİ İLE İLGİLİ TÜM SORUNLAR SİZE AİTTİR. HERHANGİ BİR HATALI BİLGİDEN DOLAYI DOĞABİLECEK OLAN BÜTÜN SERVİS, TAMİR VEYA DÜZELTME MASRAFLARI SİZE AİTTİR.

İLGİLİ KANUNUN İCBAR ETTİĞİ DURUMLAR VEYA YAZILI ANLAŞMA HARİCİNDE HERHANGİ BİR ŞEKİLDE TELİF HAKKI SAHİBİ VEYA YUKARIDA İZİN VERİLDİĞİ ŞEKİLDE KİTABI DEĞİŞTİREN VEYA YENİDEN DAĞITAN HERHANGİ BİR KİŞİ, BİLGİNİN KULLANIMI VEYA KULLANILAMAMASI (VEYA VERİ KAYBI OLUŞMASI, VERİNİN YANLIŞ HALE GELMESİ, SİZİN VEYA ÜÇÜNCÜ ŞAHISLARIN ZARARA UĞRAMASI VEYA BİLGİLERİN BAŞKA BİLGİLERLE UYUMSUZ OLMASI) YÜZÜNDEN OLUŞAN GENEL, ÖZEL, DOĞRUDAN YA DA DOLAYLI HERHANGİ BİR ZARARDAN, BÖYLE BİR TAZMİNAT TALEBİ TELİF HAKKI SAHİBİ VEYA İLGİLİ KİŞİYE BİLDİRİLMİŞ OLSA DAHİ, SORUMLU DEĞİLDİR.

## 1. Genel Bakış

Bir **düzenli ifade** (**regex** veya **şablon**) bir matematiksel dizge kümesini açıklayan bir metin dizgesidir. *s* dizgesi, *r* düzenli ifadesi tarafından açıklanan bir dizge kümesi ise *r* düzenli ifadesi *s* dizgesi ile **eşleşir**.

Regex kütüphanesini kullanarak yapabileceğiniz:

- Bir dizge belirtilen bir şablona tamamen uyarsa bunu görürsünüz.
- Bir dizge içinde belirtilen bir şablon ile uyuşan bir altdizgeyi arayabilirsiniz.

Bazı düzenli ifadeler sadece bir dizge ile eşleşir, yani bir kümenin içinde eşleşen yalnız bir üye vardır. Örneğin **f<sub>oo</sub>** düzenli ifadesi sadece **f<sub>oo</sub>** dizgesi ile eşleşir, başka bir şeyle eşleşmez.

Bazıları da bir dizgeden fazlası ile eşleşir, yani bir kümenin içinde eşleşen birden fazla üye vardır. Örneğin **f<sub>\*</sub>** düzenli ifadesi sıfır veya daha fazla sayıda **f**lerden oluşan birden fazla dizge ile eşleşebilir. Sizin de gördüğünüz gibi, düzenli ifadeler içindeki bazı karakterler kendileri ile (**f** gibi) eşleşirken, bazıları da (**\*** gibi) kendileriyle eşleşmek yerine çok sayıda farklı dizgeyi açıklayan bir kalıp olarak belirtilir.

Bir düzenli ifadeyi Regex kütüphane işlevlerinde eşleşmeleri bulmak ya da arama yapmak amacıyla kullanmadan önce bir Regex şablonu derleme işlevi ile derlemelisiniz. Bir **derlenmiş şablon** kütüphane işlevleri tarafından kullanılabilir duruma getirilmiş bir düzenli ifadedir. Bir şablonu bir kere derledikten sonra eşleşmeleri bulmak ya da arama yapmak için defalarca kullanabilirsiniz.

Regex kütüphanesi iki kaynak dosyasından oluşur: `regex.h` ve `regex.c`.

Regex, düzenli ifadeler üzerinde işlem yapabileceğiniz üç ayrı işlev grubuna sahiptir. Gruplardan biri olan GNU grubu işlevler en güçlü olanlardır ama diğerleri ile, POSIX ve Berkeley Unix grupları ile tam uyumlu değildir; arayüzü özellikle GNU için tasarlanmıştır. Diğer iki grup işlev, POSIX ve Berkeley UNIX düzenli ifade işlevlerinin arayüzleri ile benzer arayüzleri kullanırlar.

Bu kısmı düzenli ifade kullanan Emacs türü uygulamaların kullanıcılarını değil, yazılımcıları düşünerek yazdık. Regex kütüphanelerini tamamı ile tanımladık. Burada bir uygulamanın anlayacağı şekilde düzenli ifadelerin nasıl yazılacağını değil, bir bütünlük içinde Regex kütüphanesini anlattık.

## 2. Düzenli İfade Sözdizimi

**Karakterler** yazabildiğiniz şeylerdir. **İşleçler** ise bir ya da daha fazla karakterle eşleşen bir düzenli ifade içindeki şeylerdir. Düzenli ifadeleri, bir ya da daha fazla karakteri belirtmek için kullandığımız işleçlerden meydana getirebilirsiniz.

Çoğu karakter kendisiyle eşleşen işleci temsil eder, çünkü bunlar kendileriyle eşleşirler. Bu karakterlere **sıradan karakterler** deriz. Diğer karakterler ise işleçlerin hepsini ya da bazı parçalarını temsil ederler. Örneğin, **.** (nokta) karakteri her karakterle eşleşen işleç adını alır (şaşırmayın hemen hemen her şey ile eşleşir) ve bu tür karakterlere **özel karakterler** deriz. Bir karakterin ne işleci olduğunu belirleyen iki şey vardır:

1. Düzenli ifade sözdizimi,
2. Karakterin düzenli ifade içindeki durumu.

Bundan sonraki bölümde bu konu ayrıntıları ile anlatılmaktadır.

### 2.1. Sözdizimi Bitleri

Herhangi bir düzenli ifade sözdiziminde bazı karakterler daima özeldir; bazıları bazen özeldir ve diğerleri ise asla özel karakter olmazlar. Verilen bir düzenli ifadenin sözdiziminin Regex tarafından tanınması bu düzenli ifadenin şablon tamponunun **syntax** alanındaki değere bağlıdır.

Bir düzenli ifadeyi derleyerek bir şablon tamponu için bellekte yer ayırabilirsiniz. Şablon tamponlar için daha ayrıntılı bilgiyi [GNU Şablon Tamponları](#) (sayfa: 17) ve [POSIX Şablon Tamponları](#) (sayfa: 27) bölümlerinde bulabilirsiniz. Derleme hakkında daha ayrıntılı bilgiyi ise [GNU Düzenli İfadelerinin Derlenmesi](#) (sayfa: 19), [POSIX Düzenli İfadelerinin Derlenmesi](#) (sayfa: 27) ve [BSD Düzenli İfadelerinin Derlenmesi](#) (sayfa: 30) konuları altında bulabilirsiniz.

Regex, **syntax** alanının değerinin bitlerden oluştuğunu varsayar; bu bitlere **sözdizimi bitleri** denir. Pek çok durumda, hangi karakterin hangi işleci temsil ettiği konusunda etkilidirler. Bahsi geçen işleçlerin anlamlarını [Ortak İşleçler](#) (sayfa: 8), [GNU İşleçleri](#) (sayfa: 15) ve [GNU Emacs İşleçleri](#) (sayfa: 16) konularında tanımladık.

Bir başvuru kaynağı olması için, aşağıda bütün sözdizimi bitlerinin tam bir listesini alfabetik bir sırada bulabilirsiniz:

### RE\_BACKSLASH\_ESCAPE\_IN\_LISTS

Bu bit birse, bir [liste](#) (sayfa: 11) içindeki `\` kendinden sonra gelen karakteri önceler (karakter bir özel karakterse onu sıradan bir karakter yapar); sıfırsa, `\` listelerin içinde sıradan bir karakter olarak ele alınır. (`\` karakterinin listeler dışında ne yaptığı [Tersbölü Karakteri](#) (sayfa: 8) bölümünde anlatılmıştır.)

### RE\_BK\_PLUS\_QM

Bu bit birse, `\+` bir veya daha fazlası ile eşleşme işleci, `\?` ise sıfır veya daha fazlası ile eşleşme işlecinin temsil eder. Bu bit sıfırsa, `+` bir veya daha fazlası ile eşleşme işleci, `?` ise sıfır veya daha fazlası ile eşleşme işlecinin temsil eder. `RE_LIMITED_OPS` biti bir ise bu bit anlamsız olur.

### RE\_CHAR\_CLASSES

Bu bit birse, listelerde karakter sınıflarını kullanabilirsiniz, sıfırsa kullanamazsınız.

### RE\_CONTEXT\_INDEP\_ANCHORS

Bu bit birse, `^` ve `$` liste dışında her yerde özeldir. Bu bit sıfırsa, bu karakterler sadece bir liste içinde özeldir. [Satır başı ile eşleşme işleci](#) (sayfa: 14) ve [Satır sonu ile eşleşme işleci](#) (sayfa: 15) bölümlerine bakınız.

### RE\_CONTEXT\_INDEP\_OPS

Bu bit birse, bazı karakterler liste dışında her yerde özeldir. Bu bit sıfırsa, bu karakterler bazı genel durumlarda özel ve diğer heryerde sıradandırlar. Özellikle, bu bit sıfırsa `*` ve `RE_LIMITED_OPS` sözdizimi biti sıfırsa `+` ve `?` (veya `RE_BK_PLUS_QM` sözdizimi bitine bağlı olarak `\+` ve `\?`) düzenli ifade içinde ilk değillerse ya da bir VEYA işleci ya da grup başlatma işlecinin hemen ardından gelmiyorlarsa yineleme işleçlerini temsil ederler. `RE_INTERVALS` birleşmişse ve geçerli bir aralığın başlangıcı ise `{` (veya `RE_NO_BK_BRACES` bitine bağlı olarak `\{`) içinde aynı durum geçerlidir.

### RE\_CONTEXT\_INVALID\_OPS

Bu bit birse, yineleme ve VEYA işleçleri bir düzenli ifade içinde belirli konumlarda bulunamazlar. Özellikle, aşağıdaki durumlarda düzenli ifade geçersizdir:

- Bir yineleme işleci düzenli ifade içinde ilk ise veya satır başı eşleştirme, grup başlatma ya da VEYA işleçlerinden hemen sonra geliyor ise.
- Bir VEYA işleci düzenli ifade içinde ilk veya son ise, satır sonu eşleştirme işleçinden hemen önce ya da grup başlatma işleci ya da bir VEYA işlecinin hemen sonra geliyorsa.

Bu bit sıfırsa, yineleme ve veya karakterlerini temsil eden karakterleri düzenli ifade içinde her hangi bir yere koyabilirsiniz. Gerçekte, belirli konumlardaki işleçler olup olmamaları diğer sözdizimi bitlerine dayanır.

### RE\_DOT\_NEWLINE

Bu bit birse, herhangi bir karakter ile eşleştirme işleği satırsonu karakteri ile eşleşir; sıfırsa bunu yapmaz.

### RE\_DOT\_NOT\_NULL

Bu bit birse, herhangi bir karakterle eşleşme işleği boş karakterle eşleşmez. Sıfırsa eşleşir.

### RE\_INTERVALS

Bu bit birse, Regex aralık işleçlerini tanır; değilse tanımaz.

### RE\_LIMITED\_OPS

Bu bit birse, bir ya da daha fazlası ile eşleştirme, sıfır ya da biri ile eşleştirme ya da VEYA işleçleri Regex tarafından tanınmaz; değilse tanınır.

### RE\_NEWLINE\_ALT

Bu bit birse, satırsonu karakteri VEYA işleçini temsil eder, değilse satırsonu karakteri sıradan karakterlerden biri olur.

### RE\_NO\_BK\_BRACES

Bu bit birse, { aralık başlatma işleçini, } aralık kapatma işleçini temsil eder; değilse, \{ aralık başlatma işleçini, \} aralık kapatma işleçini temsil eder. Sadece RE\_INTERVALS biti bir ise bu bit anlamlıdır.

### RE\_NO\_BK\_PARENS

Bu bit birse, ( grup başlatma işleçini, ) grup kapatma işleçini temsil eder; sıfırsa, \( grup başlatma işleçini, \) grup kapatma işleçini temsil eder.

### RE\_NO\_BK\_REFS

Bu bit birse, \rakam adresleme işleği olarak tanınmaz; sıfırsa tanınır.

### RE\_NO\_BK\_VBAR

Bu bit birse, | VEYA işleçini temsil eder; sıfırsa \ | VEYA işleçini temsil eder. RE\_LIMITED\_OPS biti birse bu bit anlamsızdır.

### RE\_NO\_EMPTY\_RANGES

Bu bit birse, içinde bitiş noktası başlangıç noktasından daha küçük olan bir aralık bulunan düzenli ifade geçersizdir; sıfırsa, boş bir aralık olarak kabul edilir.

### RE\_UNMATCHED\_RIGHT\_PAREN\_ORD

Bu bit birse ve düzenli ifade grup başlatma işleğine sahip değilse, Regex, )'in (RE\_NO\_BK\_PARENS bitinin durumuna göre) grup kapatma işleci olduğunu var sayacaktır.

## 2.2. Öntanımlı Sözdizimleri

Şayet Regex ile yazılım geliştiriyorsanız, bir şablon tamponun (bkz. [GNU Şablon Tamponları](#) (sayfa: 17) ve [POSIX Şablon Tamponları](#) (sayfa: 27)) **syntax** alanını, [sözdizimi bitlerini](#) (sayfa: 4) kendiniz belirleyerek ya da Regex tarafından tanımlandığı biçimde belirtebilirsiniz. Bu yapılandırmalar, belirli yazılımlar –GNU Emacs, POSIX Awk, geleneksel Awk, Grep, Egrep– tarafından kullanılan sözdizimlerini, temel ve genişletilmiş POSIX düzenli ifadelerinin sözdizimlerine ek olarak tanımlar.

Doğrudan `regex.h` başlık dosyasından alınmış önceden tanımlı sözdizimleri:

```
#define RE_SYNTAX_EMACS 0

#define RE_SYNTAX_AWK
(RE_BACKSLASH_ESCAPE_IN_LISTS | RE_DOT_NOT_NULL
```

```

| RE_NO_BK_PARENS          | RE_NO_BK_REFS          | RE_NO_BK_VBAR          | RE_NO_EMPTY_RANGES          | RE_UNMATCHED_RIGHT_PAREN_ORD)
\

#define RE_SYNTAX_POSIX_AWK          | RE_BACKSLASH_ESCAPE_IN_LISTS)
\

#define RE_SYNTAX_GREP          | RE_CHAR_CLASSES          | RE_HAT_LISTS_NOT_NEWLINE          | RE_INTERVALS          | RE_NEWLINE_ALT)
\

#define RE_SYNTAX_EGREP          | RE_CONTEXT_INDEP_ANCHORS          | RE_CONTEXT_INDEP_OPS          | RE_HAT_LISTS_NOT_NEWLINE          | RE_NEWLINE_ALT          | RE_NO_BK_PARENS          | RE_NO_BK_VBAR)
\

#define RE_SYNTAX_POSIX_EGREP          | RE_INTERVALS          | RE_NO_BK_BRACES)
\

/* P1003.2/D11.2, section 4.20.7.1, lines 5078ff. */
#define RE_SYNTAX_ED RE_SYNTAX_POSIX_BASIC

#define RE_SYNTAX_SED RE_SYNTAX_POSIX_BASIC

/* Temel ve genişletilmiş POSIX düzenli ifade sözdizimlerinin
her ikisi için de geçerli sözdizimi bitleri. */
#define _RE_SYNTAX_POSIX_COMMON          | RE_CHAR_CLASSES          | RE_DOT_NEWLINE          | RE_DOT_NOT_NULL          | RE_INTERVALS          | RE_NO_EMPTY_RANGES)
\

#define RE_SYNTAX_POSIX_BASIC          | _RE_SYNTAX_POSIX_COMMON          | RE_BK_PLUS_QM)
\

/* Farklar: Sadece ..._POSIX_BASIC'deki RE_BK_PLUS_QM'in yerini
RE_LIMITED_OPS alır, yani, \? \+ \| artık tanınmaz. Aslında, \` gibi
diğer işleçler iptal edilmediğinden bu pek de ufak tefek değildir. */
#define RE_SYNTAX_POSIX_MINIMAL_BASIC          | _RE_SYNTAX_POSIX_COMMON          | RE_LIMITED_OPS)
\

#define RE_SYNTAX_POSIX_EXTENDED          | _RE_SYNTAX_POSIX_COMMON          | RE_CONTEXT_INDEP_ANCHORS          | RE_CONTEXT_INDEP_OPS          | RE_NO_BK_BRACES          | RE_NO_BK_PARENS          | RE_NO_BK_VBAR          | RE_UNMATCHED_RIGHT_PAREN_ORD)
\

/* Farklar: ..._POSIX_EXTENDED'deki RE_CONTEXT_INVALID_OPS'nin yerini
RE_CONTEXT_INDEP_OPS alır ve RE_NO_BK_REFS eklenir. */
#define RE_SYNTAX_POSIX_MINIMAL_EXTENDED          | _RE_SYNTAX_POSIX_COMMON          | RE_CONTEXT_INDEP_ANCHORS          | RE_CONTEXT_INVALID_OPS          | RE_NO_BK_BRACES          | RE_NO_BK_PARENS          | RE_NO_BK_REFS          | RE_NO_BK_VBAR          | RE_UNMATCHED_RIGHT_PAREN_ORD)
\

```

### 2.3. Elemanların ve Karakterlerin Alfabetik Sıralaması

POSIX bir karakterin anlamını, bir alfabetik sıralama elemanına genelleştirir ve bir **alfabetik sıralama elemanı** belirli bir alfabetik dizilim içinde bir alfabetik sıralama birimi olarak tanımlanmış bir ya da daha fazla bayttan oluşan bir dizilim olarak tanımlar.

Bu, bir karakterin anlamını iki farklı yolla geneller. İlkinde tek bir karakter iki ya da daha fazla alfabetik sıralama elemanına karşılık olabilir. Örneğin, Almanca "es–zet", sıralamada bir alfabetik sıralama elemanı **s** den sonra gelen başka alfabetik sıralama elemanı **s** olarak ele alınır. İkincisinde, iki ya da daha fazla karakter tek bir alfabetik sıralama elemanına karşılık gelebilir. Örneğin İspanyolcada, **ll** iki karakterden oluşan tek bir alfabetik sıralama elemanıdır ve **l**'den sonra, **m**'den önce gelir.

POSIX'in "alfabetik sıralama elemanı", "karakter" in gerekliliği fikrini muhafaza ettiğinden ve daha tanıdık geldiğinden bu belgede ikincisini kullandık.

### 2.4. Tersbölü Karakteri

`\` karakteri, hangi *sözdizimi bitlerinin* (sayfa: 4) belirtildiğine ve hangi bağlamda kullandığınıza göre değişen dört ayrı anlama sahiptir. 1) kendisini temsil eder, 2) sonraki karakteri önceler, 3) bir işlecin parçası olur veya 4) hiçbir şey yapmaz.

1. `RE_BACKSLASH_ESCAPE_IN_LISTS` biti sıfırsa, bir *liste* (sayfa: 11) içinde kendisini temsil eder. Örneğin, `[\ ]`, `\` ile eşleşir.
2. Kendinden sonra gelen karakteri özel bir karakterse şu hallerde sıradan yapar:
  - bir listenin dışında,<sup>(1)</sup>
  - `RE_BACKSLASH_ESCAPE_IN_LISTS` biti bir ise bir listenin içinde.
3. Belli bir sıradan karakteri öncelediğinde bir işleç olur; bazan sadece belirli sözdizimi bitleri bir ise bu olur. *Sözdizimi Bitleri* (sayfa: 4) içindeki `RE_BK_PLUS_QM`, `RE_NO_BK_BRACES`, `RE_NO_BK_VAR`, `RE_NO_BK_PARENS`, `RE_NO_BK_REF` durumlarına bakınız. Ayrıca:
  - `\b` *sözcük sınırıyla eşleşme işlecini* (sayfa: 16) temsil eder.
  - `\B` *sözcük–içi eşleşme işlecini* (sayfa: 16) temsil eder.
  - `<` *sözcük başlangıcıyla eşleşme işlecini* (sayfa: 16) temsil eder.
  - `>` *sözcük sonuyla eşleşme işlecini* (sayfa: 16) temsil eder.
  - `\w` *sözcük bileşenleriyle eşleşme işlecini* (sayfa: 16) temsil eder.
  - `\W` *sözcük bileşeni olmayanlarla eşleşme işlecini* (sayfa: 16) temsil eder.
  - `\`` *tampon başlangıcıyla eşleşme işlecini* (sayfa: 16), `\'` *tampon sonuyla eşleşme işlecini* (sayfa: 16) temsil eder.
  - şayet Regex `emacs` önişlemci sembolü ile derlenmişse, `\sınıf` *sınıfla sözdizimsel eşleşme işlecini* (sayfa: 16), `\Sınıf` *sınıfla sözdizimsel olmayan eşleşme işlecini* (sayfa: 16) temsil eder.
4. Tüm diğer durumlarda, Regex `\` karakterini yoksayar. Örneğin, `\n`, `n` ile eşleşir.

## 3. Ortak İşleçler

Düzenli ifadeleri işleçlerden oluşturursunuz. İzleyen bölümlerde, GNU'nun da kullandığı POSIX tarafından belirtilmiş düzenli ifade işleçlerini tanımlayacağız. Pek çok işleç birden fazla karakter ile temsil edilebilir. Hangi karakterin hangi koşul altında hangi işleci temsil ettiği *Düzenli İfade Sözdizimi* (sayfa: 4) bölümünde açıklanmıştır.

İki farklı şekilde temsil edilebilen pek çok işleç için, birinci yol tek bir karakter ve diğeri `` ile öncelenmiş karakterdir. Örneğin grup başlatma işlecini ya ( ya da \ ( temsil eder. Hangisinin temsil ettiği sözdizimi bitinin değerine bağlıdır, grup başlatma işleci için bakılacak sözdizimi biti `RE_NO_BK_PARENS`'dir. Peki neden iki farklı gösterim var? Bir kısmı geçmişin, bir kısmı da POSIX'in dayatmasıdır.

Son olarak, hemen hemen bütün karakterler *liste* (sayfa: 11) içerisinde özel anlamlarını kaybederler.

### 3.1. Kendisiyle Eşleşme İşleci

Bu işleç karakterin kendisi ile eşleşir. Bütün sıradan karakterler (Bkz. *Düzenli İfade Sözdizimi* (sayfa: 4)) bu işleci temsil eder. Örneğin, `f` daima sıradan bir karakterdir, bu nedenle `f` düzenli ifadesi sadece `f` dizgesi ile eşleşir. Herhangi bir sebepten dolayı `ff` ile eşleşmez.

### 3.2. Herhangi Bir Karakterle Eşleşme İşleci

Bu işleç basılabilen veya basılamayan herhangi bir tek karakter ile eşleşir. Aşağıdaki durumlar dışında:

#### satırsonu karakteri (\n)

`RE_DOT_NEWLINE` sözdizimi biti sıfırda bu karakterle eşleşmez.

#### boş karakter (\0)

`RE_DOT_NOT_NULL` sözdizimi biti bir ise bu karakterle eşleşmez.

Bu işleci `.` (nokta) karakteri temsil eder. Örneğin, `a.b` ifadesi `a` ile başlayan ve `b` ile biten herhangi bir üç karakterli dizge ile eşleşir.

### 3.3. Birleştirme İşleci

Bu işleç `a` ve `b` gibi iki düzenli ifadeyi birleştirir. Bu işleç herhangi bir karakter ile temsil edilmez; basitçe `b`'yi `a`'nın ardına yerleştirirsiniz. Sonuç, `a` ile başlayan ve `b` ile devam eden bir dizge ile eşleşen bir düzenli ifadedir. Örneğin, `xy` düzenli ifadesi sadece `xy` dizgesi ile eşleşir.

### 3.4. Yineleme İşleçleri

Yineleme işleçleri kendilerinden önce gelen düzenli ifadeyi, istenen miktarda tekrar etmeye yarar.

#### 3.4.1. Sıfır veya daha fazlası ile eşleşme işleci

Bu işleç, örneğe uygun hale getirene kadar, kendinden önce yer alan ve mümkün olan en küçük yapıdaki düzenli ifadeyi gerekli sayıda–sıfır dahil–tekrarlamaya yarar. Bu işleç, `*` ile temsil edilir. örneğin; `o*` içerisinde sıfır veya daha fazla `o` bulunan her hangi bir dizge ile eşleşir. Bu işleç mümkün olan en küçük parça üzerinde işlem yaptığı için, `fo*` içinde sadece `o` tekrarlanır, `fo` değil. Bu nedenle, `fo*` sıra ile; `f`, `fo`, `foo`, ... ile eşleştirilir.

Sıfır veya fazlası ile eşleşme işleci bir sonek olduğu için, kendinden önce bir düzenli ifade bulunmadığı durumlarda hiçbir işe yaramazlar. Bu, şu durumlarda geçerli olur:

- bir düzenli ifade içindeki ilk karakter ise,
- bir satır başı ile eşleşme işlecinin, grup başlatma işlecinin ya da VEYA işlecinin ardından kullanılmışsa.

Bu tür durumlarda üç farklı şey olabilir:

1. Şayet `RE_CONTEXT_INVALID_OPS` sözdizimi biti bir ise, düzenli ifade geçersiz olur.

2. Şayet `RE_CONTEXT_INVALID_OPS` biti sıfır ve `RE_CONTEXT_INDEP_OPS` biti bir ise `*` sıfır veya fazlası ile eşleşme işlecini temsil eder (böylece boş bir dizge üzerinde işlem yapılabilir).
3. Aksi takdirde, `*` sıradan bir karakterdir.

Eşleştirme işlemi, sıfır veya fazlası ile eşleştirme işlecinin kendinden önceki mümkün olan en küçük düzenli ifade ile öncelikle eşleştirilmesi ve bu işlemin gerektiği kadar yinelenmesi sürecidir. Daha sonra şablonun kalanı ile eşleştirmeye devam edilir.

Şayet şablonun kalanı ile eşleştirilemez ise, gerektiği kadar (kaç kere gerekliyse) geri gidilir ve işlem tekrarlanır. Her geri gidişte bir önceki eşleştirme iptal edilerek şablonun bütününe en yakın eşleşme bulunmaya çalışılır ya da hiçbir eşleşme bulunamaz. Örneğin; `ca*ar` düzenli ifadesi `caaar` dizgesi ile eşleştirilirken, eşleştirici ilk önce, dizgenin üç `a`sı ile düzenli ifadenin `a*`'ini eşleştirir. Bununla beraber, düzenli ifadenin son `ar`'i dizgenin son `r`'si ile eşleşemez. Bu durumda geriye dönülerek dizgedeki son `a` eşleşmesi iptal edilerek geriye kalan `ar` ile eşleşme sağlanır.

### 3.4.2. Bir veya daha fazlası ile eşleştirme işleci

`RE_LIMITED_OPS` sözdizimi biti bir ise, Regex bu işleci tanımaz. Aksi takdirde, `RE_BK_PLUS_QM` sözdizimi biti sıfır ise işleci `+`, değilse `\+` temsil eder.

Bu işleç *sıfır veya daha fazlası ile eşleştirme işleğine* (sayfa: 9) benzer. Tek fark bu işlecin kendinden önceki düzenli ifadeyi en az bir kere yinelenmesidir.

Örneğin; `+` nın bir veya fazlası ile eşleşme işleci olduğunu kabul edersek, `ca+ar` ifadesi `caar` ve `caaaaar` dizgeleriyle eşleşir, fakat `car` ile eşleşmez.

### 3.4.3. Sıfır veya bir kere eşleşme işleci

`RE_LIMITED_OPS` sözdizimi biti bir ise, Regex bu işleci tanımaz. Aksi takdirde, `RE_BK_PLUS_QM` sözdizimi biti sıfır ise işleci `?`, değilse `\?` temsil eder.

Bu işleç kendinden önceki düzenli ifadeyi en az bir kere yinelenmesi ya da hiç yinelenmemesi dışında *sıfır veya daha fazlası ile eşleştirme işleci* (sayfa: 9) gibidir.

Örneğin, `ca?r` düzenli ifadesi `car` ve `cr` dışında hiçbir şey ile eşleşmez.

### 3.4.4. Sınırlı sayıda yineleme işleçleri

`RE_INTERVALS` sözdizimi biti bir ise, Regex *sınırlı sayıda yineleme işleçlerini* tanıır. Bunlar, kendilerinden önceki mümkün olan en küçük düzenli ifadeyi belirli bir sayıda tekrar ederler.

`RE_NO_BK_BRACES` sözdizimi biti bir ise, `{` karakteri *sınırlı sayıda yineleme başlatma işleci*, `}` karakteri de *sınırlı sayıda yineleme bitirme işleci* olur, aksi takdirde `\{` ve `\}` işleçleri geçerli olur.

Özel olarak, sınırlı sayıda yineleme başlatma ve bitirme işleçlerini `{` ve `}` karakterlerinin temsil ettiğini varsayarsak:

`{sayı}`

Önceleyen düzenli ifadeyi tam olarak *sayı* kere eşleştirir.

`{sayı,}`

Önceleyen düzenli ifadeyi en az *sayı* kere eşleştirir.

`{sayı1, sayı2}`

Önceleyen düzenli ifadeyi en az *sayı1*, en çok *sayı2* kere eşleştirir.

Sınırlı sayıda yineleme ifadesinin geçersiz olduğu durumlar (ancak, onu içeren düzenli ifadenin geçerliliği gerekli değildir):

- *sayı1* > *sayı2*
- *sayı1*, *sayı1* veya *sayı2* sıfırla `RE_DUP_MAX` sabiti arasında bir değer değilse. (`RE_DUP_MAX` sembolik sabiti `regex.h` başlık dosyasında tanımlanmıştır.)

Sınırlı sayıda yineleme ifadesi geçersiz ve `RE_NO_BK_BRACES` biti bir ise, Regex ifade içindeki karakterlerin sıradan olduğunu varsayar; bitin değeri sıfırsa, düzenli ifade geçersizdir.

Sınırlı sayıda yineleme ifadesi geçerli fakat üzerinde işlem yapılacak bir önceleyen düzenli ifade yoksa ve `RE_CONTEXT_INVALID_OPS` biti bir ise, düzenli ifade geçersizdir; bitin değeri sıfırsa Regex ifade içindeki karakterleri — tersbölüler hariç — sıradan kabul eder.

### 3.5. VEYA İşleci

`RE_LIMITED_OPS` sözdizimi biti bir ise, Regex bu işleci tanımaz. Aksi takdirde, `RE_NO_BK_VBAR` biti bir ise, `|` değilse `\|` ile temsil edilir.

Düzenli ifade içinde bu işleç ile ayrılan düzenli ifadelerden biri ile eşleştirme yapılır. Örneğin, `|` VEYA işleci ise, `foo|bar|quux` düzenli ifadesi `foo`, `bar` veya `quux` dizgelerinden herhangi biri ile eşleşecektir.

VEYA işleci, düzenli ifadelerin mümkün olan en büyük bağlamı üzerinde işlem yaparlar. (Başka bir deyişle VEYA işleci herhangi bir düzenli ifade işlecine göre en düşük önceliğe sahiptir.) Bu nedenle argümanlarını sınırlamanın tek yolu onları gruplamaktır. Örneğin, `( ve )` gruplama işleçleri ve `|` VEYA işleci ise, `fo(o|b)ar` düzenli ifadesi, `fooar` ya da `fobar` ile eşleşecektir. (Düzenli ifade `foo|bar` ise `foo` veya `bar` ile eşleşir.)

Eşleştirme mümkün olan en büyük dizge üzerinde mümkün olan bütün olasılıkların denenmesi ile yapılır. Örneğin; `(foo|foo)*(qbarquux|bar)` düzenli ifadesi, `fooqbarquux` dizgesi ile eşleştirilirken, bu eşleşme olmaz, ilk (ve en ilk) birleşim eşleşirdi ve bu da sadece `fooqbar` olurdu.

### 3.6. Liste İşleçleri

*Listeler*, *köşeli ayrıçlı ifadeler* olarak da bilinir ve bir ya da daha fazla öğeden oluşan kümelerdir. Burada *öge* bir karakter, bir karakter sınıfı veya bir aralık ifadesidir. Bir listeye konabilecek öge türü sözdizimi bitleri ile belirlenir. Son iki öğeyi aşağıdaki iki altbölümde açıklayacağız. Boş listeler geçersizdir.

Bir *eşleşme listesi*, liste öğelerinden biri ile temsil edilen tek bir karakterle eşleşir. Bir eşleşme listesini `[` ile gösterilen bir *eşleşme listesi başlatma işleci* ile başlayan bir ya da daha fazla öge içeren ve `]` ile gösterilen bir *eşleşme listesi bitirme işleci* ile biten bir ifade olarak oluşturabilirsiniz.

Örneğin; `[ab]` ifadesi `a` veya `b` ile eşleşir. `[ad]*` ise boş bir dizge veya içinde herhangi bir sırada `a` veya `b` olan dizgelerle eşleşir. Regex, içerisinde `[` olan ama karşılığı olan `]` karakterini içermeyen düzenli ifadeleri geçersiz sayar.

Bir *eşleşmeme listesi* eşleşme listesine benzer fakat sadece, liste içerisinde temsil edilmeyen karakterleri eşleştirirler. Bir eşleşmeme listesini başlatmak için, bir eşleşme listesi başlatma işleci yerine *eşleşmeme listesi başlatma işleci* kullanırsınız (`[^(2)`).

Örneğin; `[^ab]`, `a` ve `b` hariç herhangi bir karakter ile eşleşebilir.

*Şablon tamponu* (sayfa: 17) içinde `posix_newline` alanı bir ise, eşleşmeme listesi satırsonu karakteri ile eşleşmez.

Pek çok karakter, liste içerisinde, özel anlamlarını kaybederler. Liste içerisinde özel anlamı olan karakter:

`]`

Şayet listedeki ilk karakter değilse, listeyi kapatır. Bu nedenle liste içerisinde `]` karakterini kullanmak istiyorsanız, bu karakteri en başa koymalısınız.

`\`

Şayet `RE_BACKSLASH_ESCAPE_IN_LISTS` sözdizimi biti bir ise, önündeki karakteri önceler.

`[ :`

Şayet `RE_CHAR_CLASSES` sözdizimi biti bir ise ve kendinden sonrakiler geçerli bir karakter sınıf ifadesi ise, *karakter sınıfı başlatma işleçini* (sayfa: 12) temsil eder.

`: ]`

Şayet `RE_CHAR_CLASSES` sözdizimi biti bir ise ve kendinden öncekiler geçerli bir karakter sınıf ifadesi ise, *karakter sınıfı bitirme işleçini* (sayfa: 12) temsil eder.

`-`

Şayet bir liste içerisinde ilk veya son karakter değilse ya da bir aralığın uç noktası değilse, *aralık işleci* (sayfa: 13)ni temsil eder.

Tüm diğer karakterler sıradandır. Örneğin; `[.*]` ifadesi hem `.` hem de `*` ile eşleşir.

### 3.6.1. Karakter Sınıfı İşleçleri

Şayet `RE_CHARACTER_CLASSES` sözdizimi biti bir ise, Regex, listelerin içindeki karakter sınıf ifadelerini tanır. Bir *karakter sınıfı ifadesi* belirtilen sınıftan bir karakter ile eşleşir. Bir karakter sınıfı fadesini, bir *karakter sınıfı başlatma işleci* (`[ :` ile temsil edilir) ile bir *karakter sınıfı bitirme işleci* (`: ]` ile temsil edilir) arasına koyarak oluşturabilirsiniz. Karakter sınıf isimleri ve anlamları şunlardır:

`alnum`

harfler ve rakamlar

`alpha`

harfler

`blank`

sisteme bağlıdır, GNU için boşluk ve sekme (tab) karakterleridir.

`cntrl`

kontrol karakterleri (ascii kodlamada, sekizlik tabanda 040. karakterden önceki tüm karakterler ile 0177. karakter)

`digit`

rakamlar

`graph`

boşluk karakteri hariç `print` sınıfı ile aynıdır.

`lower`

küçük harfler

`print`

basılabilir karakterler (ascii kodlamada, boşluk, `~` ve sekizlik tabanda 040'dan 0176'ya kadar tüm karakterler)

`punct`

ne kontrol ne de alfasayısal karakterler

`space`

boşluk, satırbaşı, satırsonu, düşey sekme ve sayfa ileri karakterleri

`upper`

büyük harfler

`xdigit`

onaltılık rakamlar: **0–9, a–f, A–F**

Bunlar, GNU C kütüphanesinin `ctype.h` başlık dosyasındaki tanımlamalara karşı düşer. Örneğin, `[ :alpha: ]` ifadesi standart `isalpha` işlevine karşılık gelir. Regex, karakter sınıfı ifadelerini sadece listelerin içinde tanır; bu nedenle `[ [ :alpha: ] ]` ifadesi herhangi bir harf ile eşleşirken, `[ :alpha: ]` ifadesi bir köşeli ayrıçlı ifadenin dışında olduğundan ve ardından bir yineleme işlevi gelmediğinden sadece kendisiyle eşleşir.

### 3.6.2. Aralık İşlevi

Regex bir liste içindeki **aralık ifadelerini** tanır. Kullanımdaki yerelin alfabetik sıralamasına bağlı olarak iki karakter arasında kalan bütün karakterleri temsil ederler. Bir aralık ifadesini iki karakterin arasına bir **aralık işlevi** yerleştirerek oluşturabilirsiniz<sup>(3)</sup> Aralık işlevi – karakteri ile temsil edilir. Örneğin, bir liste içerisindeki **a–f** ifadesi, **a**'dan **f**'ye kadar olan bütün karakterleri kapsar.

`RE_NO_EMPTY_RANGES` sözdizimi biti bir ise ve aralığın bitiş karakteri alfabetik sıralamada başlangıç karakterinden küçükse aralık ifadesi geçersizdir. Örneğin, `[ z–a ]` düzenli ifadesi geçersiz olurdu. Bu bit sıfırsa, Regex böyle bir aralığı boş kabul eder.

– karakteri aralık işlevini temsil ettiğinden bu karakterin kendisini bir liste ögesi yapmak isterseniz aşağıdaki yöntemlerden birini kullanmalısınız:

- – karakterini listenin ya başına ya da sonuna yerleştirin.
- Alfabetik sıralamada – karakterinden küçük bir karakterle başlayan veya bu karakterle ya da daha büyük bir karakterle biten bir aralığın içinde bırakın. Bir aralık, bir listenin ilk ögesi olmadıkça, bir – karakteri bir aralığın başlangıç karakteri olamaz ancak, bitiş karakteri olabilir. Regex, bir – karakteri başka bir – karakteri ile öncelenmedikçe, – karakterini aralık işlevi olarak ele alır. Örneğin `ascii` kodlamada, `)`, `*`, `+`, `,`, `-`, `.` ve `/` karakterleri sıralamada peşpeşe gelirler. `[ ]+---/` ifadesine bakınca, iki aralığın olduğunu düşünebilirsiniz: `)++` ve `--/`. Halbuki, `)++` ve `+++` aralıkları ile `/` karakterini kapsar. Yani ifade `,` ile eşleşir ama `.` ile eşleşmez.
- Liste içerisine başlangıç noktası – olan bir aralığı ilk öge olarak yerleştirin.

Örneğin, `[ -a–z ]` ifadesi herhangi bir küçük harf veya bir – karakteri ile eşleşir.

### 3.7. Gruplama İşleçleri

Bir **grup** aynı zamanda bir **alt ifade**dir ve bir **grup başlatma işlevi** ile başlatılıp, bir takım başka işleçlerden sonra bir **grup sonlandırma işlevi** ile sonlandırılır. Regex böyle bir ifadeyi yazılım geliştirme dillerindeki ve matematiksel ifadelerdeki gibi tek bir birim olarak ele alır.

Bu nedenle, **grupları** kullanarak:

- Bir **VEYA işlevinin** (sayfa: 11) veya bir **yineleme işlevinin** (sayfa: 9) argümanlarını sınırlarsınız.
- Verilen grup ile eşleşen bir alt dizgenin indislerinin izini sürebilirsiniz. Daha ayrıntılı bir açıklama için **Yazmaçların Kullanımı** (sayfa: 23) bölümüne bakınız. Bu sizin:
  - **adres işlevini** (sayfa: 14)
  - **yazmaçları** (sayfa: 23)

kullanmanızı sağlar.

Eğer `RE_NO_BK_PARENS` sözdizimi biti bir ise, `(` karakteri grup başlatma işlecini, `)` karakteri de grup sonlandırma işlecini temsil eder; aksi takdirde bunlar, `\(` ve `\)` ile temsil edilir.

Eğer `RE_UNMATCHED_RIGHT_PAREN_ORD` sözdizimi biti bir ise ve grup sonlandırma işleci için bir grup başlatma işleci yoksa, Regex onu `)` ile (yani kendisi ile) eşleştirir.

### 3.8. Grup Adresleme İşleci

Şayet, `RE_NO_BK_REF` sözdizimi biti bir ise, Regex adreslemeleri tanır. Bir adres, evvelce belirtilmiş bir grup ile eşleştirilir. Grup adresleme işleci, bir düzenli ifadenin *rakam*'ıncı *grubundan* (sayfa: 13) sonra herhangi bir yere yerleştirilmiş bir `\rakam` ile temsil edilir.

*rakam*, 1'den 9'a kadar bir rakam olmalıdır. Eşleştirici, saptadığı ilk 9 gruba bu numaraları atar. `\1`'den `\9`'a kadar herhangi bir işleci, kendisine karşı düşen grubun grup sonlandırma işlecinden sonra kullanarak, grubun eşleştiği alt dizge ile işleşen bir grup gibi kullanılabilir. Yani bir grubu tekrar tekrar yazmak yerine sadece grup adresleme işlecini kullanmak yeterli olur.

Grup adresleme işleçleri aşağıdaki kurallara göre eşleşirler (aşağıdaki örneklerde `(` karakteri grup başlatma işleci, `)` karakteri grup sonlandırma işleci, `{` karakteri sınırlı sayıda yineleme başlatma işleci ve `}` karakteri sınırlı sayıda yineleme sonlandırma işleci olarak kullanılmıştır.):

- Şayet grup bir alt dizge ile eşleşiyorsa, adres benzer bir alt dizge ile eşleşir. Örneğin; `(a)\1` ifadesi `aa` ile eşleşirken `(bana)na\1bo\1` ifadesi `bananabanabobana` ile eşleşir. Aynı şekilde, `(.*)\1` ifadesi (`RE_DOT_NEWLINE` sözdizimi biti sıfır ise satırsonu karakteri ile eşleşmez) herhangi bir iki aynı parçadan oluşan dizge ile eşleşir; `(.*)` dizgenin ilk bölümü ile `\1` ise ikinci bölümü ile eşleşir.
- Şayet grup defalarca eşleştiriliyorsa (arkasında bir yineleme işleci olabilir), adres grupla en son eşleşen alt dizge ile eşleşir. Örneğin; `((a*)b)*\1\2` ifadesi `aabababa` ile eşleşir; ilk grup 1 (dıştaki olan), `aab` ile ve grup 2 (içteki), `aa` ile eşleşir. Daha sonra grup 1 `ab` ile ve grup 2 `a` ile eşleştirilir. Böylece, `\1` parçası `ab` ile ve `\2` parçası `a` ile eşleşmiş olur.
- Şayet grup bir eşleşmeye katılmıyorsa, Örneğin: eşleşmeyen bir VEYA işlecinin bir parçası veya sıfır kere yineleme yapan bir yineleme işlecinin bir parçası olabilir, bu durumda adres bütün eşleşmelerde başarısız olur. Örneğin, `(one()|two())-and-(three\2|four\3)` ifadesi `one-and-three` ve `two-and-four` ile eşleşir, fakat asla `one-and-four` veya `two-and-three` ile eşleşmez. Örneğin ifade, `one-and-` ile eşleşirse 2. grup boş dizge ile eşleşirken 3. grup eşleşmeye konu olmaz. Bu durumda `four` ile eşleşme olsa bile, 3.grup ile eşleşme arandığında (çünkü ifadede `\3`, `four` ile birlikte verilmiştir), 3. grup bir eşleşmeye konu olmadığından `four` ile eşleşme başarısız olacaktır.

Bir grup adresleme işleci bir yineleme işlecine argüman olarak kullanılabilir. Örneğin; `(a(b))\2*` ifadesi, `a` ve `a`'dan sonra gelen iki veya daha fazla `b` ile eşleşir. Benzer şekilde, `(a(b))\2{3}` ifadesi `abbbb` ile eşleşir.

Bir *rakam*'ıncı grup yoksa düzenli ifade geçersiz olur.

### 3.9. Demirleme İşleçleri

Bu tür işleçler bir şablonu, bir dizgenin sadece başlangıcı veya bitimi ile ya da bir satırın başlangıcı veya bitimi ile eşleşmeye zorlarlar.

#### 3.9.1. Satır başı ile eşleşme işleci

Bu işleç, bir satırsonu karakterini izleyen boş bir dizge ile eşleşebilir. Bu durumda, şablon satırın başına *demirlemiştir* denebilir.

Aşağıda açıklanan durumlarda, `^` karakteri bu işleci temsil eder. Diğer durumlarda, `^` sıradandır.

- `^foo` örneğindeki gibi `^`, şablondaki ilk karakter ise.
- `RE_CONTEXT_INDEP_ANCHORS` sözdizimi biti bir ise ve bir köşeli ayraçlı ifade dışında ise.
- `a\(^b\)` ve `a|^b` örneklerindeki gibi, bir *grup başlatma işlecinden* (sayfa: 13) ya da bir *VEYA işlecinden* (sayfa: 11) hemen sonra geliyorsa.

Bu kurallar eşleştirelemeyen `^` içeren geçerli şablonlara uygulanır; örneğin, `RE_CONTEXT_INDEP_ANCHORS` sözdizimi biti bir ise `foo^bar` ifadesindeki `^` satır başı ile eşleşme işleci olarak değerlendirilir.

*Şablon tamponunda* (sayfa: 17) `not_bol` alanı bir ise, `^` işleci dizgenin başlangıcı ile eşleştirilirken başarısız olur. Bunu kullanışlı buluyorsanız, *POSIX Eşleştirilmesi* (sayfa: 28) bölümüne bakınız.

*Şablon tamponunda* (sayfa: 17) `newline_anchor` alanı bir ise, `^` işleci bir satırsonu karakterinden sonrasına eşleştirilirken başarısız olur. Bu, içinde satırsonu karakteri bulunan hatalı satırların gözardı edilmesini istediğiniz durumlarda yararlıdır.

### 3.9.2. Satır sonu ile eşleşme işleci

Bu işleç, bir satırsonu karakterinden önceki boş bir dizge ile eşleşebilir. Bu durumda, şablon satırın sonuna *demirlemiştir* denebilir.

Daima `$` ile temsil edilir. Örneğin, `foo$` ifadesi tek başına `foo` dizgesi ile ya da örneğin `foo\nbar` dizgesinin ilk üç karakteri ile eşleşir.

Sözdizimi bitleri ve şablon tampon alanları ile etkileşimi, tam olarak, `^` işlecinin tersidir; önceki bölüme göz atınız ("ilk" yerine "son", "sonra" yerine "önce" ve "başlama" yerine "sonlandırma" getirerek).

## 4. GNU İşleçleri

Bu kısımdaki işleçler GNU tarafından tanımlanmıştır (ve POSIX ile uyumsuzdur).

### 4.1. Sözcük İşleçleri

Bu bölümdeki işleçler Regex'in sözcük parçalarını tanıması ile ilgilidir. Regex, bir karakterin bir sözcüğün parçası olup olmadığını anlamak için bir sözdizimi tablosu kullanır, yani bir karakterin bir sözcüğün ögesi olup olmadığını bakar.

#### 4.1.1. Emacs–dışı Sözdizimi Tabloları

Bir *sözdizimi tablosu*, kullandığınız karakter kümesindeki karakterlere göre indislenmiş bir dizidir. Bu nedenle, ASCII kodlamada, bir sözdizimi tablosu 256 adet eleman içerir. Regex, sözdizimi tablosu olarak `char *` türünde olan `re_syntax_table` değişkenini kullanır. Bazı durumlarda değişkeni kendisi ilkendir, bazı durumlarda da sizin ilkendireceğinizi umar.

- Regex, `emacs` ve `SYNTAX_TABLE` önışlemci sembolleri tanımsız (`#undefine`) olarak derlenmişse, `re_syntax_table` değişkeni için bellek ayırır ve bir `i` elemanı ile ilkendir. Bu ilkendirme `i` elemanı bir harf, rakam ya da `_` ise  `sword` değilse sıfır değeri ile yapılır.
- Regex, `emacs` tanımsız ancak `SYNTAX_TABLE` tanımlı olarak derlenmişse, Regex, geçerli bir sözdizimi tablosu olarak `re_syntax_table` değişkenini `char *` türünde sizin tanımlayacağınızı umar.
- Regex'in `emacs` önışlemci sembolünün tanımlanarak derlenmesi ile ilgili durum *Emacs Sözdizimi Tabloları* (sayfa: 16) bölümünde açıklanmıştır.

#### 4.1.2. Sözcük sınırlarıyla eşleşme işleci (\b)

Bu işleç (\b ile temsil edilir) bir sözcüğün başındaki ya da sonundaki boş dizge ile eşleşir. Örneğin, \brat\b ifadesi tek başına bir sözcük olarak **rat** ile eşleşir.

#### 4.1.3. Sözcük-İçi eşleşme işleci (\B)

Bu işleç (\B ile temsil edilir) bir sözcüğün içindeki boş dizge ile eşleşir. Örneğin, c\Brat\Be ifadesi **crate** ile eşleşirken, dirty \Brat ifadesi **dirty rat** dizgesiyle eşleşmez.

#### 4.1.4. Sözcük başlangıcıyla eşleşme işleci (\<)

Bu işleç (\< ile temsil edilir) bir sözcüğün başındaki boş dizge ile eşleşir.

#### 4.1.5. Sözcük sonuyla eşleşme işleci (\>)

Bu işleç (\> ile temsil edilir) bir sözcüğün sonundaki boş dizge ile eşleşir.

#### 4.1.6. Sözcük bileşenleriyle eşleşme işleci (\w)

Bu işleç (\w ile temsil edilir) bir sözcüğün ögesi olan herhangi bir karakter ile eşleşir.

#### 4.1.7. Sözcük bileşeni olmayanlarla eşleşme işleci (\W)

Bu işleç (\W ile temsil edilir) bir sözcüğün ögesi olmayan herhangi bir karakter ile eşleşir.

### 4.2. Tampon İşleçleri

Bu bölümdeki işleçler tamponlarla çalışır. Emacs için bir **tampon** doğal olarak bir Emacs tamponudur. Diğer uygulamalar için ise Regex, eşleştirilecek dizgenin tamamının tampon olduğunu varsayar.

#### 4.2.1. Tampon başlangıcıyla eşleşme işleci (\ `)

Bu işleç (\ ` ile temsil edilir) tamponun başlangıcındaki boş dizge ile eşleşir.

#### 4.2.2. Tampon sonuyla eşleşme işleci (\')

Bu işleç (\' ile temsil edilir) tamponun sonundaki boş dizge ile eşleşir.

## 5. GNU Emacs İşleçleri

Aşağıdaki işleçler Regex'in sadece **emacs** önişlemci sembolü tanımlanarak derlenmesi halinde kullanılabileceğiniz GNU tarafından tanımlanmış ancak POSIX uyumlu olmayan işleçlerdir.

### 5.1. Sözdizimsel Sınıf İşleçleri

Bu bölümdeki işleçler, Regex'in karakterlerin sözdizimsel sınıflarını tanımasına ihtiyaç duyarlar. Regex bunları tanıyabilmek için bir sözdizimi tablosu kullanır.

#### 5.1.1. Emacs Sözdizimi Tabloları

Bir **sözdizimi tablosu**, sizin karakter kümenizdeki karakterler tarafından indislenmiş bir dizidir. Bu nedenle, ASCII kodlama sisteminde, bir sözdizimi tablosu 256 adet elemana sahiptir.

Şayet Regex **emacs** önışlemci sembolü tanımlanarak derlenmişse, Regex sizin **re\_syntax\_table** değişkenini bir Emacs sözdizimi tablosu olarak tanımlamanızı ve bu tabloyu ilklendirmenizi bekler. Emacs sözdizimi tabloları *Regex'inkinden* (sayfa: 15) bile daha karmaşıktırlar. Emacs'ın sözdizimi tablolarının tarifi için *Emacs Sözdizimi Tabloları* (sayfa: 16) bölümüne bakınız.

### 5.1.2. Sözdizimsel Sınıfları Eşleştirme İşleç

Bu işleç, sözdizimsel sınıfı belirli bir karakter ile temsil edilen herhangi bir karakteri eşleştirebilir. **\ssınıf** şeklinde kullanılır. Buradaki *sınıf* istediğiniz sözdizimsel sınıfı temsil eden karakterdir. Örneğin; **w**, sözcük bileşeni olan karakterlerinin sözdizimsel sınıfını temsil eder. Bu nedenle **\sw** herhangi bir sözcük bileşeni karakter ile eşleşebilir.

### 5.1.3. Sözdizimsel Olmayan Sınıfları Eşleştirme İşleç

Bu işleç Sözdizimsel sınıfları eşleştirme işleğine benzerdir ama tek farkı sözdizimsel sınıfın özel bir karakter ile temsil edilmediği durumlarda eşleştirme yapmasıdır. Bu işleç **\Ssınıf** ile temsil edilir. Örneğin, **w** sözcük bileşeni karakterlerin sözdizimsel sınıfını temsil eder, böylece **\Sw** sözcük bileşeni olmayan herhangi bir karakter ile eşleşir.

## 6. Ne Eşleştirilir?

Regex, genellikle, dizgeleri "en soldaki en uzun" kuralına göre eşleştirir; yani, en soldaki en uzun eşleşmeyi seçer. Tabii ki bu alt ifadeler içeren bir düzenli ifadenin basitçe her alt ifade için soldan sağa en uzun eşleştirmeyi seçtiği anlamına gelmez. Ana düzenli ifade içindeki genel eşleşmenin mümkün olan en uzun eşleşme olması da gereklidir.

Örneğin: **(ac\*)(c\*d[ac]\*)\1** ifadesi, **acdacaaa** ile eşleşir, ilk alt ifade içindeki en uzun eşleşme olan **acdac** ile değil.

## 7. Regex ile Yazılım Geliştirme

Bu bölümde Regex veri yapılarını ve işlevlerini C yazılımları içerisinde nasıl kullanacağınız açıklanmaktadır. Regex üç tane arayüze sahiptir: biri GNU için tasarlanmıştır, biri POSIX uyumludur ve öteki Berkeley UNIX uyumlu olanıdır.

### 7.1. GNU Regex İşlevleri

Şayet POSIX veya Berkeley UNIX ile uyumlu olmak zorunda olmayan kodlar yazıyorsanız, bu işlevleri kullanabilirsiniz. Bunlar diğer arayüzlerden daha fazla seçeneğe sahiptirler. (Ç.N. — Özellikle eşleşmeleri arayacağınız dizge Türkçe'ye özgü karakterler içeriyorsa, bu arayüzü kullanmak kaçınılmazdır.)

#### 7.1.1. GNU Şablon Tamponları

Verilen bir düzenli ifadenin derlenmesi, eşleştirilmesi veya araştırılması için bir şablon tamponu yaratmak zorundasınızdır. Bir **şablon tamponu** derlenmiş bir düzenli ifadeyi tutar.<sup>(4)</sup>

Aynı anda birden çok ve farklı şablon tamponuna sahip olabilirsiniz, her biri değişik bir düzenli ifade için derlenmiş bir şablonu tutabilir.

```
struct re_pattern_buffer
```

veri türü

`regex.h` şablon tamponu yapısını aşağıda belirtilen şekilde tanımlar:

`unsigned char *buffer`

Derlenmiş şablonu tutan tampon. `unsigned char *` şeklinde bildirilir, çünkü elemanları bazen dizi indisleri olarak kullanılır.

`unsigned long allocated`

Tampon için ayrılan baytların sayısı.

`unsigned long used`

Tampon içinde kullanılmış bayt sayısı.

`reg_syntax_t syntax`

Şablonun birlikte derlendiği sözdizimi ayarları.

`char *fastmap`

Varsa bir hızlı eşleme gösterici, aksi takdirde sıfır. Eşleşmelerin imkansız başlangıç noktalarını atlamak için varsa `fastmap`, `re_search` işlevi tarafından kullanılır.

`char *translate`

Karşılaştırma öncesi tüm karakterlere uygulanacak çeviri tablosu, çeviri yoksa sıfır. Çeviri, derlenirken şablona ve eşleştirilirken dizgeye uygulanır.

`size_t re_nsub`

Derleyici tarafından bulunan alt ifadelerin sayısı.

`unsigned can_be_null`

Bu şablon boş dizgeyi eşleştiremezse sıfır yoksa bir. Aslında sadece, `fastmap` kullanmamızın gerekip gerekmeyeceğini saptamak için `re_search_2` işlevi tarafından kullanılır. Bu nedenle bunu mükemmel olarak ayarlayamadık; *Hızlı Eşlemlerle Arama* (sayfa: 22) bölümüne bakınız.

`unsigned regs_allocated`

`REGS_UNALLOCATED` tanımlıysa, `max (RE_NREGS, re_nsub + 1)` grup için `regs` yapısı içinde yer ayrılır. `REGS_REALLOCATE` tanımlıysa, gerekliyse yer yeniden ayrılır. `REGS_FIXED` tanımlıysa, olan kullanılır; bu öntanımlıdır.

```
#define REGS_UNALLOCATED 0
#define REGS_REALLOCATE 1
#define REGS_FIXED 2
```

`unsigned fastmap_accurate`

`regex_compile` işlevi ile bir şablonu derlediğiniz zaman sıfır olur; şayet `fastmap`, `re_compile_fastmap` işlevi ile güncellenirse bir olur.

`unsigned no_sub`

Birse, `re_match_2` işlevi alt ifadeler hakkında bilgi döndürmez.

`unsigned not_bol`

Bir ise satır başı ile eşleşme işlevi dizgenin başlangıcıyla eşleşmez. Öntanımlı değeri 1'dir.

`unsigned not_eol`

Satır sonu ile eşleşme işlevi için `not_bol` alanına benzer.

`unsigned newline_anchor`

Birse, demir, satırsonu karakteri ile eşleşir. Öntanımlı değeri 1'dir.

### 7.1.2. GNU Düzenli İfadesinin Derlenmesi

GNU'da, belirtilmiş olan bir düzenli ifade için hem eşleme hem de arama yapabilirsiniz. Bunu yapmak için, öncelikle onu bir *şablon tamponu* (sayfa: 17) içinde derlemelisiniz.

**re\_syntax\_options**

değişken

Düzenli ifadeler birlikte derlendikleri söz dizimlerine göre eşleşirler; GNU'da, **re\_compile\_pattern** derleme işlevi çağrılmadan önce, **re\_syntax\_options** değişkeni (`regex.h` içinde bildirilmiş ve `regex.c` içinde tanımlanmıştır) ile hangi sözdizimini istediğinizi belirtebilirsiniz. *Sözdizimi Bitleri* (sayfa: 4) ve *Öntanımlı Sözdizimleri* (sayfa: 6) bölümlerine bakınız.

**re\_syntax\_options** değişkeninin değerini istediğiniz zaman değiştirebilirsiniz. Ancak, genellikle, değeri bir kere atanır ve bir daha değiştirilmez.

**re\_compile\_pattern** işlevi argüman olarak **struct re\_pattern\_buffer** türünde bir *şablon tamponu* alır. Aşağıdaki alanları iklendirmelisiniz:

`translate`

Bir çeviri tablosunu göstermesini istiyorsanız bir ile, istemiyorsanız sıfır ile iklendirin. Çeviri tabloları, *GNU Çeviri Tabloları* (sayfa: 22) bölümünde açıklanmıştır.

`fastmap`

Bir hızlı eşlem istiyorsanız bir ile istemiyorsanız sıfır ile iklendirin.

`buffer`

`allocated`

**re\_compile\_pattern** işlevinin derlenmiş şablon için bellek ayırmasını istiyorsanız bunların her ikisini de sıfır ile iklendirin. Daha önce **malloc** ile ayırdığınız bir bellek bloğu varsa ve Regex'in bunu kullanmasını istiyorsanız, adresi ile **buffer**'ı genişliği ile de **allocated**'i iklendirin:

```
struct re_pattern_buffer buf;

buf.allocated = 1;
buf.buffer = xmalloc (buf.allocated);
```

**re\_compile\_pattern** işlevi, gerektiği takdirde ayrılan bloğu genişletmek için **realloc** işlevini kullanacaktır.

```
char *re_compile_pattern(const char *düzenli_ifade,          işlev
                        const int   ifade_boyutu,
                        struct re_pattern_buffer *şablon_tamponu)
```

Bir şablon tamponunu derlemek için kullanılır.

*düzenli\_ifade* düzenli ifadenin adresidir, *ifade\_boyutu* ifadenin uzunluğu ve *şablon\_tamponu* da şablon tamponunun adresidir.

**re\_compile\_pattern** işlevi şablon tamponunu başarıyla derlerse sıfır ile döner ve *şablon\_tamponu* derlenen tamponu gösterir. Şablon tamponunun işlev tarafından ayarlanan alanları:

`buffer`

derlenen şablon ile iklendirilir.

`used`

*şablon\_tamponu* içindeki derlenmiş şablonun bayt cinsinden uzunluğu ile iklenir.

`syntax`

`re_syntax_options` değişkeninin o anki değeri ile iklenir.

`re_nsub`

*düzenli\_ifade* içindeki alt ifadelerin sayısı ile iklenir.

`fastmap_accurate`

Teorik olarak *şablon\_tamponu* içine derlediğiniz şablon bir önceki derlemeden farklı olacağından sıfırla iklenir; bu durumda (bir derlenmiş şablon olmaksızın bir hızlı eşlem yapamayacağınızdan), `fastmap` ya uyumsuz bir hızlı eşlem içerecekti ya da hiçbir şey.

`re_compile_pattern` işlevi *düzenli\_ifade*'yi, derleyemezse, *POSIX Düzenli İfadelerinin Derlenmesi* (sayfa: 27) bölümünde listesi verilen hatalardan birine karşılık olan bir hata dizgesi ile döner.

### 7.1.3. GNU Eşleştirme İşlevi

GNU yöntemi ile eşleştirmenin anlamı, belirttiğiniz bir yerden başlayarak bir dizgenin mümkün olan en çok karakteri ile eşleştirme yapmaya çalışmaktır.

```
int re_match(struct re_pattern_buffer *şablon_tamponu,           işlev
             const char *dizge,
             const int boyut,
             const int başlangıç,
             struct re_registers *yazmaçlar)
```

Bir şablonu bir *şablon\_tamponuna derledikten* (sayfa: 19) sonra bu işlevi kullanarak bir dizgenin bu şablonla eşleştirilmesini isteyebilirsiniz.

*şablon\_tamponu* derlenmiş düzenli ifadeyi içeren tamponunun adresidir. *dizge* eşleştirilmesini istediğiniz dizgedir ve satırsonu karakteri ile boş karakter içerebilir. *boyut* bu dizgenin uzunluğudur. *başlangıç* eşleşmenin aranacağı dizge indisidir; *dizge* içindeki ilk karakterin indisi sıfırdır. Açıklamasını *Yazmaçların Kullanımı* (sayfa: 23) bölümünde bulacağınız *yazmaçlar* için güvenle sıfır değerini aktarabilirsiniz.

`re_match` işlevi, *şablon\_tamponu* içindeki düzenli ifadeyi, *dizge* dizgesi ile *şablon\_tamponu*'nun `syntax` alanı içindeki sözdizimine göre eşleştirir. (`syntax` alanının nasıl iklendirildiğini öğrenmek için *GNU Düzenli İfadesinin Derlenmesi* (sayfa: 19) bölümüne bakınız.)

İşlev, *dizge* dizgesinin hiçbir parçası ile eşleşme bulamazsa -1 ile, bir iç hata oluşursa -2 ile, aksi takdirde *dizge* içinde şablonla eşleşen karakterlerin sayısı (sıfır olabilir) ile döner.

Örnek: *şablon\_tamponu*'nun `a*` ifadesi için derlenmiş şablonu içerdiğini, *dizge*'nin `aaaaab` dizgesini gösterdiğini (burada *boyut* değeri 6 olmalıdır) varsayalım. *başlangıç* 2 ise, `re_match` 3 değeri ile döner, yani, `a*` ifadesi *dizge* içindeki son 3 `a` ile eşleşmiş olur. *başlangıç* 0 olsaydı dönen değer 5 olacaktı, yani, `a*` ifadesi *dizge* içindeki bütün `a`larla eşleşecekti. *başlangıç* 5 ya da 6 olsaydı işlev 0 ile dönecekti.

Şayet *başlangıç* sıfırdan küçük ya da *boyut*'dan büyük verilirse işlev -1 ile döner.

### 7.1.4. GNU Arama İşlevi

*Arama* bir dizge içindeki ardışık konumlarda eşleşmenin başlangıcının aranmasıdır. `re_search` işlevi bunu yapar.

**re\_search** işlevini çağırmadan önce *düzenli ifadenizi derlemeniz* (sayfa: 19) gerekir.

```
int re_search(struct re_pattern_buffer *şablon_tamponu,           işlev
              const char              *dizge,
              const int                boyut,
              const int                başlangıç,
              const int                aralık,
              struct re_registers      *yazmaçlar)
```

*aralık* argümanı dışında tüm argümanlar **re\_match** (sayfa: 20) işlevi ile aynıdır.

*aralık* pozitifse, **re\_search** ilk eşleşmeyi *başlangıç* ile belirtilen indiste arar, başarısız olursa *başlangıç* + 1'e geçer ve böyle ilerleyerek *başlangıç* + *aralık*'a kadar işlem tekrarlanır. *aralık* negatifse, **re\_search** ilk eşleşmeyi *başlangıç* ile belirtilen indiste arar, başarısız olursa *başlangıç* - 1'e geçer ve böyle gider.

Şayet *başlangıç* sıfırdan küçük ya da *boyut*'dan büyük verilirse işlev -1 ile döner. *aralık* pozitifse, **re\_search** *aralık* değerini gerekirse, *başlangıç* + *aralık* - 1, sıfır ile *boyut* arasında olacak şekilde ayarlar; bu durumda *dizge* dışında arama yapılmamış olur. Benzer şekilde, *aralık* negatifse, **re\_search** *aralık* değerini gerekirse, *başlangıç* + *aralık* + 1, sıfır ile *boyut* arasında olacak şekilde ayarlar.

Şayet *şablon tamponu*'nun **fastmap** alanı sıfırsa, **re\_search** eşleştirme işlemini ardarda gelen konumlarda başlatır; aksi takdirde aramayı daha verimli kılmak için **fastmap**'i kullanır (Bakınız, *Hızlı Eşlemlerle Arama* (sayfa: 22)).

Bir eşleşme bulunmazsa, **re\_search** -1 ile döner. Bulunursa, eşleşmenin başladığı yerin indisi ile döner. Bir iç hata oluşmuşsa -2 ile döner.

### 7.1.5. Veriyi Bölerek Arama ve Eşleştirme

**re\_match\_2** ve **re\_search\_2** işlevlerini kullanarak, iki ayrı dizgeye bölünmüş veriler içinde eşleştirme ve arama yapabilirsiniz.

```
int re_match_2(struct re_pattern_buffer *şablon_tamponu,           işlev
               const char              *dizge1,
               const int                boyut1,
               const char              *dizge2,
               const int                boyut2,
               const int                başlangıç,
               struct re_registers      *yazmaçlar,
               const int                son)
```

**re\_match\_2** işlevi iki veri dizgesi ve bunların boyutları ile eşleştiricinin eşleştirme denemesi yapmasını istemedikleriniz için bir son indisi belirtilmesi dışında **re\_match** (sayfa: 20) işlevine benzer. **re\_match** işlevinde olduğu gibi, başarı durumunda dizgenin eşleşen karakter sayısı ile döner. *başlangıç* ve *son* argümanları belirtildiğinde ve *yazmaçlar*'ın içeriği kullanıldığında *dizge1* ve *dizge2* birleşik sayılır; **re\_match\_2** asla *boyut1* + *boyut2*'den daha büyük bir değerle dönmaz.

```
int re_search_2(struct re_pattern_buffer *şablon_tamponu,           işlev
                const char              *dizge1,
                const int                boyut1,
                const char              *dizge2,
                const int                boyut2,
                const int                başlangıç,
                const int                aralık,
                struct re_registers      *yazmaçlar,
                const int                son)
```

`re_search` işlevine benzer.

### 7.1.6. Hızlı Eşlemlerle Arama

Uzun bir dizge üzerinde arama yapıyorsanız bir hızlı eşlem kullanmalısınız. Bu olmaksızın, arayıcı dizge içinde ardarda her konumda eşleşme dener. Genellikle, dizge içindeki karakterlerin pek çoğu bir eşlemeyi başlatamaz. Dizge içindeki belirli bir noktadan eşleme yapmayı denemek, karakterin bir eşlemeyi başlatıp başlatamayacağını bir tablodan kontrol etmekten çok daha fazla vakte mal olur. İşte hızlı eşlem böyle bir tablodur.

Daha belirgin olarak, bir hızlı eşlem karakter kümenizdeki karakterlerle indislenmiş bir dizidir. Bundan dolayı, `ascii` karakter kodlaması altında bir hızlı eşlem 256 elemanlıdır. Arayıcının belirtilmiş bir şablon tamponu ile bir hızlı eşlemi kullanmasını isterseniz, bellekte dizi için yer ayırmalı ve dizinin adresini şablon tamponunun `fastmap` alanına atamalısınız:

```
#define BYTEWIDTH 8

struct re_pattern_buffer compiled;
char fastmap[1 << BYTEWIDTH];

compiled.fastmap = fastmap;
```

Ya hızlı eşlemi siz derlersiniz ya da `re_search` bunu sizin için yapar; `fastmap` alanının değeri sıfırdan farklı ise özel olarak derlenmiş bir şablonu kullanarak yapacağınız ilk arama sırasında hızlı eşlem `re_search` tarafından derlenir.

```
int re_compile_fastmap(struct re_pattern_buffer *şablon_tamponu) işlev
```

Bir hızlı eşlemi kendiniz derlemek isterseniz bu işlevi kullanmalısınız. `şablon_tamponu` bir şablon tamponunun adresidir. `c` karakteri şablon için bir eşleşme başlatmalysa, `re_compile_fastmap`, `şablon_tamponu->fastmap[c]` değerini sıfırdan farklı yapar. Hızlı eşlem derlenebilmişse işlev sıfır değeriyle, bir iç hata oluşmuşsa `-2` değeriyle döner. Örneğin, `|` bir VEYA işlevi ise ve `şablon_tamponu, a|b` ifadesi için derlenmiş şablonu içeriyorsa, `re_compile_fastmap` işlevi sadece `fastmap['a']` ve `fastmap['b']` değerlerini sıfırdan farklı yapar.

`re_search` dizge içinde hareket ederken hızlı eşlem kullanırsa, dizge içindeki karakterleri hızlı eşlem içindeki karakterlerden birini buluncaya kadar ilerler ve bu karakterden itibaren eşleştirmeyi dener. Eğer eşleşme olmazsa işlemi tekrarlar. Böylece bir hızlı eşlem kullanılarak dizge içinde zaten eşleşmeyecek konumlarda `re_search` eşleştirme için zaman harcamaz.

`re_search` işlevinin hızlı eşlem kullanmasını istemeseniz, işlevi çağırılmadan önce şablon tamponunun `fastmap` alanına sıfır atayın.

Bir şablon tamponunun `fastmap` alanını bir kere ilklendirdikten sonra, bir daha bunu yapmaya ihtiyacınız olmaz — içinde yeni bir şablon derleseniz bile — yeter ki, alan sizin bir hızlı eşlem isteyip istemediğinizine tepki vermeye ayarlı olsun.

### 7.1.7. GNU Çeviri Tabloları

Bir şablon tamponunun `translate` alanına bir çeviri tablosu yerleştirirseniz, GNU Regex işlevleri bu şablon tamponunu düzenli ifadenin tamamına ve aranan dizge karakterlerine basit bir dönüşüm uygulamakta kullanır.

Bir `çeviri tablosu` karakter kümenizdeki karakterlerle indislenmiş bir dizidir. Bu nedenle, `ascii` karakter kümesinde bir çeviri tablosu 256 elamanlıdır. Dizinin elamanları ayrıca karakter kümenizdeki karakterlerdir. Regex işlevleri bir `c` karakteri gördüğünde onun yerine `translate[c]` kullanır, bir istisna dışında: karakter bir `\` ile öncelenmişse çeviri yapılmaz. Böylece, `\B` ve `\b` gibi işleçlerin seçilebilirliği garanti altına alınır.

Örneğin; bütün küçük harfleri büyük harflere dönüştürmek üzere tasarlanmış bir tablo, eşleştiricinin harf büyüklükleri farkını yok saymasına sebebiyet verir.<sup>(5)</sup> Böyle bir tablo, küçük harfler dışındaki karakterleri kendileriyle, küçük harfleri de karşılıkları olan büyük harflerle eşleyecektir. Ascii kodlama altında, böyle bir tabloyu nasıl ilklendireceğiniz aşağıda gösterilmiştir (tablonun ismi **case\_fold**'dur):

```
for (i = 0; i < 256; i++)
    case_fold[i] = i;
for (i = 'a'; i <= 'z'; i++)
    case_fold[i] = i - ('a' - 'A');
```

Regex'in bir şablon tamponu üzerinde bir çeviri tablosunu kullanmasını isterseniz, tablonun adresini tamponun **translate** alanına atamalısınız. Regex'in herhangi bir çeviri yapmasını istemezseniz, bu alana sıfır yerleştirin. Şayet tablonun içeriğini şablon tamponunu derleme, hızlı eşleme derleme, şablon tamponu ile eşleşme ve aramalar arasında herhangi bir zamanda değiştirmeye kalkarsanız tuhaf sonuçlar alırsınız.

### 7.1.8. Yazmaçların Kullanımı

Bir düzenli ifade içindeki bir grup, düzenli ifadenin tamamen eşleştiği bir dizgenin alt dizgesi (muhtemelen boş) ile eşleşebilir. Eşleştirici, her bir grupla eşleşen alt dizgelerin başlangıç ve bitiş noktalarını hatırlar.

Onların nelerle eşleştiğini bulmak için bir GNU *eşleştirme* (sayfa: 20) ve *arama* (sayfa: 20) işlevinin *yazmaçlar* argümanına sıfırdan farklı bir değer, örneğin `regex.h` dosyasında tanımlı olan aşağıdaki gibi bir yapının adresini atayın:

```
struct re_registers{ veri türü
    unsigned num_regs;
    regoff_t *start;
    regoff_t *end;
};
```

*num\_regs*'inci eleman hariç (aşağıya bakınız), **start** ve **end** dizilerinin *i*'nci elemanı, şablondaki *i*'nci grup hakkındaki bilgileri kaydeder. (Hiçbir C derleyicisi sıfır uzunluklu dizileri kabul etmediğinden bu diziler gösterici olarak bildirilmiştir; kavramsal olarak, onları dizi olarak düşünmek en kolaydır.)

**start** ve **end** dizileri için, eşleştiriciye aktarılan şablon tamponu içindeki **regs\_allocated** alanının değerine bağlı olarak çeşitli yöntemlerle bellek ayrılabilir.

Burada uygulanması en basit ve belki de en akıllıca yöntem; eşleştiriciye, düzenli ifade içindeki bütün grupların bilgilerinin kaydedilebileceği alanı (yeniden) ayırmaktır. **regs\_allocated** alanının değeri `REGS_UNALLOCATED` ise, eşleştirici `1 + re_nsub` yer tahsis eder (*şablon tamponu* (sayfa: 17) içindeki bir alan). Ek elemana `-1` ve **regs\_allocated** alanına `REGS_REALLOCATE` değeri atanır. Daha sonradan aynı şablon tamponu ve *yazmaçlar* argümanı ile yapılan çağrılarda, eşleştirici gerektiği kadar alanı yeniden tahsis eder.

**regs\_allocated** alanını **re\_registers** yapısının bir parçası yapmak, şablon tamponun bir parçası yapmaktan daha akıllıca bir yöntem olabilir. Fakat bu durumda, çağrıya aktarılmadan önce yapının ilklendirilmesi zorunlu olacaktır. Mevcut kodların pek çoğu bu ilklendirmeyi yapmaz ve bundan kaçınmak her halükarda en iyisidir.

**re\_compile\_pattern**, **regs\_allocated** alanına `REGS_UNALLOCATED` atar, böylece GNU düzenli ifade işlevlerini kullanırsanız, bu davranışı öntanımlı olarak elde etmiş olursunuz.

Diğer yandan, POSIX, farklı bir arayüze ihtiyaç duyar: çağrı, eşleştirici tarafından doldurulacak sabit uzunluktaki bir dizi aktarılacağını varsayar. Bu nedenle, şayet **regs\_allocated** alanının değeri `REGS_FIXED` ise, eşleştirici o diziye doldurur.

Aşağıdaki örneklerde, **re\_registers** yapısına bilgi kaydedilmesi gösterilmiştir. (Hepsinde, ( işlecinin grup başlatma, ) işlecinin ise grup sonlandırma işleci olduğu varsayılmıştır. *dizge* dizgesinin ilk karakterinin indisi 0 dır.)

- Şayet düzenli ifade, *dizge* dizgesinin bir alt dizgesi ile eşleşen ve bir alt grup içermeyen bir *i*'inci grup içeriyorsa, işlev, *yazmaçlar*->start [*i*] alanını *dizge* içindeki, *i*'inci gruba eşleşen alt-dizgenin başlangıcına, *yazmaçlar*->end [*i*] alanını da alt dizgenin sonuna denk gelen indise ayarlar. İşlev, *yazmaçlar*->start [0] ve *yazmaçlar*->end [0] alanlarını ise şablonun tamamı hakkındaki benzer bilgilere ayarlar.

Örneğin, **( (a) (b) )** ifadesini **ab** dizgesi ile eşleştirirseniz şunları elde edersiniz:

- *yazmaçlar*->start [0] için 0, *yazmaçlar*->end [0] için 2
  - *yazmaçlar*->start [1] için 0, *yazmaçlar*->end [1] için 2
  - *yazmaçlar*->start [2] için 0, *yazmaçlar*->end [2] için 1
  - *yazmaçlar*->start [3] için 1, *yazmaçlar*->end [3] için 2
- Şayet bir grup bir kereden fazla eşleşiyorsa (ardından bir yineleme işleci geliyor olabilir), işlev son eşleşen grup hakkındaki bilgileri raporlar.

Örneğin, **(a)\*** ifadesini **aa** dizgesi ile eşleştirirseniz şunları elde edersiniz:

- *yazmaçlar*->start [0] için 0, *yazmaçlar*->end [0] için 2
  - *yazmaçlar*->start [1] için 1, *yazmaçlar*->end [1] için 2
- Şayet *i*'inci grup başarılı bir eşleşmede yer almamış ise (örneğin; kullanılmamış bir VEYA işleci veya sıfır kere tekrarlamaya ayarlı bir yineleme işleci olabilir), işlev *yazmaçlar*->start [1] ve *yazmaçlar*->end [1] için -1 atar.

Örneğin, **(a)\*b** ifadesini **b** dizgesi ile eşleştirirseniz şunları elde edersiniz:

- *yazmaçlar*->start [0] için 0, *yazmaçlar*->end [0] için 1
  - *yazmaçlar*->start [1] için -1, *yazmaçlar*->end [1] için -1
- Şayet *i*'inci grup sıfır uzunluktaki bir dizge ile eşleşirse, işlev *yazmaçlar*->start [0] ve, *yazmaçlar*->end [0] alanlarını sıfır uzunluktaki dizgenin hemen sonrasındaki indise ayarlar.

Örneğin, **(a\*)b** ifadesini **b** dizgesi ile eşleştirirseniz şunları elde edersiniz:

- *yazmaçlar*->start [0] için 0, *yazmaçlar*->end [0] için 1
  - *yazmaçlar*->start [0] için 0, *yazmaçlar*->end [0] için 0
- Şayet *i*'inci grup, grup içindeki başka bir grup tarafından ihtiva edilmeyen bir *j*'inci gruba sahipse ve işlev *i*'inci grubun eşleşmesini rapor ediyorsa, *j*'inci grubun son eşleşmesi (şayet böyle bir eşleşme olmuş ise) *yazmaçlar*->start [*j*] ve *yazmaçlar*->end [*j*] içine kaydedilir.

Örneğin, **( (a\*)b )\*** ifadesi **abb** dizgesi ile ve 2. grup bir boş dizge ile eşleşirse, evvelce yapılan eşleşme ne ise onu elde etmiş oluruz:

- *yazmaçlar*->start [0] için 0, *yazmaçlar*->end [0] için 3
- *yazmaçlar*->start [1] için 2, *yazmaçlar*->end [1] için 3
- *yazmaçlar*->start [2] için 2, *yazmaçlar*->end [2] için 2

( **(a)\*b** ) \* ifadesi **abb** dizgesi ile ve son eşleşmede 2. grup eşleşmezse, şunları elde ederiz:

- *yazmaçlar*->start [0] için 0, *yazmaçlar*->end [0] için 3
- *yazmaçlar*->start [1] için 2, *yazmaçlar*->end [1] için 3
- *yazmaçlar*->start [2] için 0, *yazmaçlar*->end [2] için 1
- Şayet *i*'inci grup, grup içindeki başka bir grup tarafından ihtiva edilmeyen bir *j*'inci gruba sahipse ve işlev *yazmaçlar*->start [*i*] ve *yazmaçlar*->end [*i*] alanların -1'e ayarlamışsa, *yazmaçlar*->start [*j*] ve *yazmaçlar*->end [*j*] alanları da -1'e ayarlanır.

Örneğin, ( **(a)\*b** ) \* **c** ifadesini **c** dizgesi ile eşleştirirseniz şunları elde edersiniz:

- *yazmaçlar*->start [0] için 0, *yazmaçlar*->end [0] için 1
- *yazmaçlar*->start [1] için -1, *yazmaçlar*->end [1] için -1
- *yazmaçlar*->start [2] için -1, *yazmaçlar*->end [2] için -1

### 7.1.9. GNU Şablon Tamponlarının Serbest Bırakılması

Bir şablon tamponunun alanlarına ayrılan yeri serbest bırakmak için, **re\_pattern\_buffer** türü, POSIX şablon tamponlarının türü olan **regex\_t** türü ile eşdeğerde olduğundan, *POSIX Şablon Tamponlarının Serbest Bırakılması* (sayfa: 30) bölümünde anlatılan posix işlevini kullanabilirsiniz. Bir şablon tamponu serbest bıraktıktan sonra, bir düzenli ifadeyi eşleme veya arama işlevlerine aktarmadan önce ifadeyi tekrar *derlemeniz* (sayfa: 19) gerekir.

### 7.1.10. Bir Arama ve Değişirme Örneği

Bu örnek, bu kitapçığın özgün kopyasında bulunmamaktadır. Tam bir uygulama örneği olarak yararlı olabileceği düşünülerek bu çeviriye eklenmiştir.

Örnek kodun yaptığı iş bir dizgeyi <b>, </b> ve <i>, </i> etiketlerinden arındırmaktır. İşlemi hızlandırmak için özel bir hızlı eşlem (fastmap) kullanılmıştır. Bu özel fastmap sayesinde eşleşme araması "/<>bi" dizgesindeki karakterlerden birine rastlandığında başlatılacağından işlemin beklenenden daha çabuk biteceği düşünülmüştür. **SET\_FASTMAP ()** makrosu bu hızlı eşlem dizisini ilklendirmek içindir.

**unformat** işlevi argümanı olan *string* dizgesini bu etiketlerden arındırır ve bunu ek bir tampon kullanmadan doğrudan bu dizge üzerinde yapar. Arama işleminin her yinelenişinde dizge giderek etiketlerden arındığından ve özel bir hızlı eşlem kullanıldığından, dizge içinde aramanın başlatılacağı konumun ilettilmesine gerek duyulmamıştır.

```
void
get_regerror(int errcode, re_pattern_buffer *compiled, char *func) {

    size_t length = regerror (errcode, compiled, NULL, 0);
    char *buffer = xmalloc (length);

    regerror (errcode, compiled, buffer, length);
    fprintf(stderr, "%s: %s\n", func, buffer);
    free(buffer);
}

#define SET_FASTMAP() \
{ \
    unsigned this_char; \
    \
}
```

```

memset (fastmap, invert, (1 << BYTEWIDTH)); \
\
for (this_char = 0; this_char < strlen (fastmap_string); this_char++)\
    fastmap[fastmap_string[this_char]] = !invert; \
fastmap['\n'] = match_newline; \
}

void
unformat (char *string) {
    char *pattern;
    struct re_pattern_buffer compiled;
    char fastmap[1 << BYTEWIDTH];
    const char *comperr;
    char *fastmap_string;
    unsigned invert, match_newline;
    struct re_registers *matches;
    int execerr, len, prelen, postlen, slen;

    re_set_syntax (RE_NO_BK_PARENS | RE_NO_BK_VBAR);
    pattern = "(<.>|</.>";
    invert = 0;
    match_newline = 0;
    fastmap_string = "/<>bi";
    SET_FASTMAP ();

    memset (&compiled, 0, sizeof (compiled));
    comperr = re_compile_pattern(pattern, strlen(pattern), &compiled);
    if (comperr)
        get_regerror((int)*comperr, &compiled, "re_compile_pattern");

    /* Şablon derlendi. Biçim arıtmasına başlayabiliriz. */
    slen = strlen(string);
    compiled.fastmap = fastmap;
    matches = alloca (sizeof(matches) * (compiled.re_nsub + 1));
    while (1) {
        memset (matches, 0, sizeof (matches));
        execerr = re_search (&compiled, string, slen, 0, slen, matches);
        if (execerr == - 1) {
            /* puts ("Eşleşme bulunamadı"); */
            break;
        }
        if (execerr >= 0) {
            len = matches->end[0] - matches->start[0];
            postlen = slen - matches->end[0];
            prelen = matches->start[0];
            strncpy(string + prelen, string + matches->end[0], postlen);
            slen = prelen + postlen;
            string[slen] = '\0';
        } else {
            get_regerror(comperr, &compiled, "re_search");
        }
    }
}

```

## 7.2. POSIX Regex işlevleri

Şayet POSIX uyumlu kodlar yazıyorsanız, bu işlemlere ihtiyacınız olacaktır. Bunların arayüzleri POSIX taslağı 1003.2/D11.2 de tanımlanmıştır.

### 7.2.1. POSIX Şablon Tamponları

Belirtilen bir düzenli ifadeyi POSIX tarzında derlemek veya eşleştirmek için, tıpkı GNU için yaptığımız gibi bir [şablon tamponu](#) (sayfa: 17) sağlamamız gerekir. GNU şablon tamponlarının türü olan `re_pattern_buffer` türü, POSIX şablon tamponlarının türü olan `regex_t` türü ile eşdeğerdedir.

### 7.2.2. POSIX Düzenli İfadelerinin Derlenmesi

POSIX ile, belirtilen bir düzenli ifade için sadece arama yapabilirsiniz; onu eşleştiremezsiniz. Bunu yapabilmek için, `regcomp` kullanarak, düzenli ifadeyi bir şablon tamponu içinde derlemelisiniz.

```
int regcomp(regex_t *şablon,                               işlem
             const char *düzenli_ifade,
             int derleme_imleri)
```

Bir POSIX şablon tamponunu derlemek için kullanılır.

`şablon`, ilklendirilmiş şablon tamponunun adresi; `düzenli_ifade`, düzenli ifadenin adresi ve `derleme_imleri` ise bir bit koleksiyonu olarak derleme imleridir. Burada geçerli bitler `regex.h` başlık dosyasında tanımlanmıştır:

#### REG\_EXTENDED

POSIX Genişletilmiş Düzenli İfade sözdiziminin kullanılacağını belirtir; şayet bu bit bir ise POSIX Genişletilmiş Düzenli İfade sözdiziminin kullanılacağını, aksi takdirde POSIX Temel Düzenli İfade sözdiziminin kullanılacağını belirtilmiş olur. `regcomp` işlevi `şablon`'un `syntax` alanını buna göre düzenler.

#### REG\_ICASE

Büyük/küçük harf ayrımı yapılmaz; `regcomp` işlevi, `şablon`'un `translate` alanını, büyük/küçük harf duyarsız bir çeviri tablosuna ayarlar, burada daha önceden bulunan herşeyi değiştirir.

#### REG\_NOSUB

`şablon`'un `no_sub` alanını ayarlar; bunun ne anlama geldiğini öğrenmek için [POSIX Eşleştirmesi](#) (sayfa: 28) bölümüne bakınız.

#### REG\_NEWLINE

- [Herhangi bir karakterle eşleşme işlecine](#) (sayfa: 9), satırsonu karakteri ile eşleşmemesini söyler.
- Bir satırsonu karakteri içermeyen [eşleşmeme listesini](#) (sayfa: 11) bir satırsonu karakteri ile eşleştirir.
- [Satır başı ile eşleşme işleci](#) (sayfa: 14), `REG_NOTBOL` (sayfa: 28) bitinin durumuna bağlı olmaksızın, bir satırsonu karakterini izleyen boş bir dizge ile eşleşir.
- [Satır sonu ile eşleşme işleci](#) (sayfa: 15), `REG_NOTEOL` (sayfa: 28) bitinin durumuna bağlı olmaksızın, bir satırsonu karakterinden hemen önceki boş bir dizge ile eşleşir.

`regcomp` işlevi düzenli ifadeyi derleyebilirse, derlenmiş şablonu `*şablon`'a yerleştirir ve sıfır değeriyle döner. `syntax` alanı hariç (yukarıda açıklandığı gibi ayarlanır). Ayrıca, [GNU derleme işlevinin](#) (sayfa: 19) yaptığı benzer bir yolla aynı alanları ayarlar.

**regcomp** işlevi düzenli ifadeyi derleyemezse, aşağıda listelenen hata kodlarından biriyle döner. (Aksi belirtilmedikçe, aşağıdaki tüm örneklerin sözdizimi temel düzenli ifade sözdizimidir.)

REG\_BADRP

Örneğin, **a\*\*** içindeki ardışık yineleme işleçleri **\*\*** geçersizdir. Başka bir örnek olarak; şayet sözdizimi, genişletilmiş düzenli ifade sözdizimi ise, **\*** içinde yapacak hiçbir şeyi olmayan yineleme işleci **\*** geçersizdir.

REG\_BADBR

Örneğin, **a\{-1** içindeki **-1** tekrar sayısı olarak geçersizdir.

REG\_EBRACE

Örneğin, **a\{1** ifadesinde sınırlı sayıda yineleme bitirme işleci eksiktir.

REG\_EBRACK

Örneğin, **[a** ifadesine eşleşme listesi bitirme işleci eksiktir.

REG\_ERANGE

Örneğin, **[z-a]** aralığında aralık sonu olan **z** harf sıralaması bakımından başlangıç olan **a**'dan küçük yapılmıştır ve aralık ifadesi geçersizdir. Ayrıca, **[[:alpha:]-]** aralığında, aralığın başlangıcı olarak **[[:alpha:]]** karakter sınıfıyla aralık yine geçersizdir.

REG\_ETYPE

Örneğin, **[[:foo:]]** içindeki **foo** karakter sınıfı ismi geçersizdir.

REG\_EPAREN

Örneğin, **a)** ifadesinde grup başlatma işleci ve **\(a** ifadesinde grup sonlandırma işleci eksiktir.

REG\_ESUBREG

Örneğin, **\(a)\2** ifadesinde grup adresleme işleci **\2** olmayan bir grubu gösterdiğinden geçersizdir.

REG\_EEND

Düzenli ifade özel bir hata koduna sebep olmuyorsa döner.

REG\_EESCAPE

Örneğin, **a\** ifadesi içindeki **\** tıpkı **\** ifadesindeki gibi geçersizdir.

REG\_BADPAT

Örneğin, genişletilmiş düzenli ifade sözdiziminde, **a())b** ifadesi içindeki boş grup **()** geçersizdir.

REG\_ESIZE

Bir düzenli ifade 65536 bayttan daha büyük bir şablon tamponu gerektiriyorsa döner.

REG\_ESPACE

Bir düzenli ifade bellek taşmasına sebep oluyorsa döner.

### 7.2.3. POSIX Eşleştirmesi

POSIX tarzı eşleştirme, bir boş karakter sonlandırılmalı dizgeyi ilk karakterinden başlayarak eşleştirmek demektir. Bir ifadeyi bir *şablon tamponu içine derledikten* (sayfa: 27) sonra aşağıdaki işlevi kullanarak şablonu bir dizge ile eşleştirmeyi deneyebilirsiniz.

```
int regexec(const regex_t *şablon,                                     işlev
             const char *dizge,
             size_t eşleşen_sayısı,
             regmatch_t eşleşenler[],
             int icra_imleri)
```

*şablon*, ifadenin derlendiği şablon tamponunun adresi, *dizge* ise eşleştirilecek olan dizgedir.

*eşleşenler* ile ilgili olarak daha ayrıntılı bilgiyi *Bayt Konumlarının kullanımı* (sayfa: 29) bölümünde bulabilirsiniz. *eşleşen\_sayısı* olarak sıfır aktarırsanız ya da *şablon*'u `REG_NOSUB` derleme bitini 1 yaparak derlemişseniz, `regex` işlevi *eşleşenler*'i yoksayar; aksi takdirde, onu en az *eşleşen\_sayısı* elemanla iklendirmelisiniz. `regex` işlevi *eşleşen\_sayısı* bayt konumunu *eşleşenler* dizisine kaydeder, kullanılmayan elemanlara da *eşleşenler*[*eşleşen\_sayısı* -1]'e kadar -1 değerini atar.

*icra\_imleri* ile *çalıştırma\_imleri* belirtilir (`regex.h` dosyasında tanımlanmış olan `REG_NOTBOL` ve `REG_NOTEOL` bitleri). `REG_NOTBOL` biti bir ise, *satır başı ile eşleşme işlevi* (sayfa: 14) daima eşleştirmede başarısız olur. Bu, size satırın bir parçası ile eşleşme yapma imkanı sağlar. Bu özelliğe, şayet bir satır içinde belirtilmiş bir şablonun tekrarlanan örneklerini arıyorsanız ihtiyacınız olacaktır. Şablonlar için, satır başı ile eşleşme işlevi ile veya onsuz, bu özellik düzgün bir biçimde çalışacaktır. `REG_NOTEOL` ise *satır sonu eşleşme işlevi* (sayfa: 15) için benzer şekilde çalışacaktır; bu bit simetri için vardır.

`regex` işlevi, *şablon*'un **syntax** alanı içindeki sözdizimine bağlı olarak *dizge* içinde *şablon* için bir eşleşme bulmaya çalışır. (Bunun nasıl ayarlandığını anlamak için *POSIX Düzenli İfadelerinin Derlenmesi* (sayfa: 27) bölümüne bakınız.) Şayet derlenmiş şablon *dizge* ile eşleşir ise işlev sıfır ile döner, eşleşmezse `REG_NOMATCH` (`regex.h` dosyasında tanımlanmıştır) ile döner.

#### 7.2.4. Hataların Bildirilmesi

`regcomp` veya `regex` başarısız olursa, sıfırdan farklı bir değer olarak bir hata koduyla döner. Bu kodlar `regex.h` dosyasında tanımlanmıştır. Bu kodların anlamları için *POSIX Düzenli İfadelerinin Derlenmesi* (sayfa: 27) ve *POSIX Eşleştirilmesi* (sayfa: 28) bölümlerine bakınız. Bu kodlara ilişkin hata dizgelerini elde etmek için aşağıdaki işlevi kullanabilirsiniz.

```
int regerror(int          hata kodu,          işlev
              const regex_t *şablon,
              char         *hata_tamponu,
              size_t       boyut)
```

*hata kodu* bir hata kodudur, *şablon* hatayı oluşturan şablon tamponunun adresi, *hata\_tamponu* hata tamponu ve *boyut* da *hata\_tamponu*'nun boyudur.

`regerror` işlevi *hata kodu*na karşılık olan hata dizgesinin uzunluğu (sonlandırıcı boş karakter dahil) ile döner. Ayrıca, *boyut* ve *hata\_tamponu* sıfırdan farklıysa, *hata\_tamponu* içinde sonlandırıcı boş karakter içererek, hata dizgesinin ilk *boyut* -1 karakteri ile döner. *boyut*, *hata\_tamponu*'nun uzunluğuna eşit ya da küçük negatif olmayan bir sayı olmalıdır.

`regerror` işlevini hata dizgesi için ne kadar yer gerektiğini öğrenmek için boş bir *hata\_tamponu* ve sıfır değerli *boyut* ile çağırabilirsiniz.

#### 7.2.5. Bayt Konumlarının kullanımı

POSIX'de `regmatch_t` türündeki değişkenler benzer bilgileri tutmalarına rağmen *GNU'nun yazmaçları* (sayfa: 23) ile aynı değillerdir. POSIX'deki yazmaçlar hakkında bilgi almak için, `regex` işlevine `regmatch_t` türünde sıfırdan farklı bir *eşleşenler* adresi aktarmalısınız. `regmatch_t` yapısı `regex.h` başlık dosyasında tanımlanmıştır:

```
typedef struct{                               veri türü
    regoff_t rm_so;
    regoff_t rm_eo;
} regmatch_t;
```

Eşleştirme işlevlerinin yazmaçlar içinde bilgileri nasıl sakladığını [Yazmaçların Kullanımı](#) (sayfa: 23) bölümünden okurken, *yazmaçlar* yerine *eşleşenler*, *yazmaçlar*->*start[i]* yerine *eşleşenler[i]*->*rm\_so* ve *yazmaçlar*->*end[i]* yerine *eşleşenler[i]*->*rm\_eo* koyun.

### 7.2.6. POSIX Şablon Tamponlarının Serbest Bırakılması

Bir şablon tamponunun alanlarına ayrılan yeri serbest bırakmak için aşağıdaki işlevi kullanabilirsiniz.

```
void regfree(regex_t *şablon)
```

işlev

*şablon* serbest bırakacağınız alanlarla ayrılmış şablon tamponudur. **regfree** işlevi ayrıca, *şablon*'un **ayrılmış** ve **kullanılmış** alanlarına sıfır atar. Bir şablon tamponunu boşalttıktan sonra, [eşleştirme işlevini](#) (sayfa: 28) kullanmadan önce, bu tampon içinde tekrar bir düzenli ifade [derlemelisiniz](#) (sayfa: 27).

### 7.3. BSD Regex İşlevleri

Şayet, Berkeley UNIX ile uyumlu kodlar yazıyorsanız, arayüzleri Berkeley UNIX ile aynı olan bu işlevlere ihtiyaç duyacaksınız demektir.

#### 7.3.1. BSD Düzenli İfadelerinin Derlenmesi

Berkeley UNIX ile, sadece belirtilen bir düzenli ifade için arama yapabilirsiniz, eşleştirme yapamazsınız. Arama yapmak için, ilk önce onu derlemelisiniz. Derlemeden önce, düzenli ifade sözdiziminin **re\_syntax\_options** ([bazı sözdizimleri](#) (sayfa: 4) `regex.h` içinde belirtilmiştir) değişkeninin ayarlarına göre derlenmesini istediğinizi belirtmelisiniz. Bir düzenli ifadeyi derlemek için aşağıdaki işlevi kullanabilirsiniz.

```
char *re_comp(char *düzifd)
```

işlev

*düzifd* boş karakter sonlandırmalı düzenli ifadenin adresidir. **re\_comp** işlevi, sadece son derlenen şablon tamponunun kullanılabilceği bir dahili şablon tamponu kullanır. Bunun anlamı; şayet daha önceden derlediğiniz bir düzenli ifadeyi kullanmak istiyorsanız —fakat bu ifade sizin en son derlediğiniz olmayacak— onu tekrar derlemek zorundasınız. Şayet **re\_comp** işlevini bir boş gösterici (bir boş dizge değil) ile çağırırsanız şablon tamponun içeriği değişmez.

**re\_comp** düzenli ifadeyi derleyebilirse, sıfırla döner. Derleyemezse, bir hata dizgesi döndürür. **re\_comp** işlevinin hata dizgeleri **re\_compile\_pattern** (sayfa: 19)'inkiler ile aynıdır.

#### 7.3.2. BSD Araması

Berkeley UNIX tarzı arama, bir dizgenin ilk karakterinden itibaren aramaya başlamak ve eşleşecek ardışık pozisyonlara ulaşmaya çalışmak demektir. **re\_exec** (sayfa: 30) kullanarak bir şablonu derledikten sonra, Regex'e bu şablonu bir dizge içinde aramak için aşağıdaki işlevi kullanabilirsiniz.

```
int re_exec(char *dizge)
```

işlev

*dizge* arama yapılacak boş karakter sonlandırmalı dizgenin adresidir.

**re\_exec** başarılı olursa 1, aksi takdirde 0 ile döner. Bir [GNU hızlı işlemi](#) (sayfa: 22) özdevinimli kullanılır.

## A. Bu Kılavuzun Kopyalanması

GNU Özgür Belgeleme Lisansı ile lisanslanmış belgelerin bu lisansı içermesi gerektiğinden ve bu lisans kendisinin değiştirilmesine izin vermediğinden (buna tercüme de dahildir) lisans hiçbir değişiklik yapılmaksızın burada belgeye eklenmiştir.

(Ç.N. – GNU Özgür Belgeleme Lisansı bu özelliği sebebiyle dili İngilizce olmayan belgelerde kullanmak için uygun değildir; Türkçe belgenize İngilizce bir metin eklemek istemezsiniz, herhalde. Daha özgür –kendinin belgeye eklenmesini zorunlu kılmayan– lisanslar da var. Örneğin "Creative Commons Share Alike" kendinin belgeye eklenmesini zorunlu kılmaması dışında GNU ÖBL'ye hemen hemen eşdeğerdir.)

## GNU Free Documentation License

Version 1.2, November 2002

Copyright © 2000, 2001, 2002 Free Software Foundation, Inc.  
59 Temple Place, Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies  
of this license document, but changing it is not allowed.

### 1. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

### 2. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the

Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain `ascii` without markup, `Texinfo` input format, `LaTeX` input format, `SGML` or `XML` using a publicly available `DTD`, and standard-conforming simple `HTML`, `PostScript` or `PDF` designed for human modification. Examples of transparent image formats include `PNG`, `XCF` and `JPG`. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, `SGML` or `XML` for which the `DTD` and/or processing tools are not generally available, and the machine-generated `HTML`, `PostScript` or `PDF` produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

### 3. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the

copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

#### 4. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

#### 5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## 6. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

## 7. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## 8. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

## 9. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

## 10. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

## 11. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

## ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C) year your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU
Free Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

## Dizin

### Semboller

(	13
)	13
*	9
+	10
-	11
.	9
:]	12
[:	12
[^	11
\	8, 11
\'	16
\(	13
\)	13
\+	10
\<	16
\>	16
\?	10
\B	16
\S	17
\W	16
\'	16
\b	16
\s	17
\w	16
\{	10
\	11
\}	10
]	11
^	11, 14
	11

### A

allocated	ilklandirmesi	19
alt	ifadeler	13
Awk		6

### B

buffer	ilklandirmesi	19
--------	---------------	----

### Ç

çapalar		14, 15
---------	--	--------

### D

demirleme		14
düzenli ifadeler		
sözdizimi		4

### E

Egrep		6
Emacs		6
eşleşme listesi		11
eşleşmeme listesi		11

### F

fastmap	ilklandirmesi	19
---------	---------------	----

### G

gerileterek işlem		10, 11
GNU işlevleri ile arama		20
GNU işlevleri ile eşleştirme		20
Grep		6
grup başlatma işleci ve ^		15
gruplama		13
grupların adreslenmesi		14

### H

harf büyüklüğünün gözardı edilmesi		27
hızlı eşlemler		22

### K

karakter sınıfları		12
köşeli ayrıçlı ifade		11

### M

malloc		19
--------	--	----

### P

parantezleme		13
POSIX Awk		6

### R

regcomp	işlevi	27
regerror	işlevi	29
regex.c		4
regex.h		4
regexexec	işlevi	28
regfree	işlevi	30
regmatch_t	türü	29
regs_allocated	alanı	23
REGS_FIXED		23
REGS_REALLOCATE		23
REGS_UNALLOCATED		23
REG_EXTENDED		27
REG_ICASE		27
REG_NEWLINE		27
REG_NOSUB		27

RE\_BACKSLASH\_ESCAPE\_IN\_LIST.....5  
 RE\_BK\_PLUS\_QM .....5  
 RE\_CHAR\_CLASSES .....5  
 re\_comp işlevi.....30  
 re\_compile\_fastmap işlevi.....22  
 re\_compile\_pattern işlevi.....19  
 RE\_CONTEXT\_INDEP\_ANCHORS .....5  
 RE\_CONTEXT\_INDEP\_ANCHORS ve ^ .....15  
 RE\_CONTEXT\_INDEP\_OPS .....5  
 RE\_CONTEXT\_INVALID\_OPS .....5  
 RE\_DOT\_NEWLINE .....5  
 RE\_DOT\_NOT\_NULL .....6  
 re\_exec işlevi.....30  
 RE\_INTERVALS .....6  
 RE\_LIMITED\_OPS .....6  
 re\_match işlevi.....20  
 re\_match\_2 işlevi.....21  
 RE\_NEWLINE\_ALT .....6  
 RE\_NO\_BK\_BRACES .....6  
 RE\_NO\_BK\_PARENS .....6  
 RE\_NO\_BK\_REFS .....6  
 RE\_NO\_BK\_VBAR .....6  
 RE\_NO\_EMPTY\_RANGES .....6  
 re\_pattern\_buffer tanımı.....17  
 re\_pattern\_buffer türü .....17  
 re\_registers türü .....23  
 re\_search işlevi .....21  
 re\_search\_2 işlevi.....21  
 re\_syntax\_options değişkeni.....19  
 re\_syntax\_options iklendirmesi .....19  
 RE\_UNMATCHED\_RIGHT\_PAREN\_ORD .....6

**S**

satır başı işleci .....14  
 satır sonu işleci .....15  
 satırsonu karakteri ile eşleşme .....11  
 sınırlı sayıda yinleme işleci .....10  
 sözcük sınırlarıyla eşleşme .....16  
 sözdizimi bitleri .....4  
 sözdizimi iklendirmesi .....19  
 struct re\_pattern\_buffer tanımı .....17

**Ş**

şablon tamponları  
 tanımlanması .....17  
 şablon tamponu  
 buffer alanı  
 re\_compile\_pattern ile iklendirilmesi19  
 fastmap\_accurate alanı  
 re\_compile\_pattern ile iklendirilmesi20  
 newline\_anchor alanı .....15

not\_bol alanı .....15  
 re\_nsub alanı  
 re\_compile\_pattern ile iklendirilmesi20  
 syntax alanı  
 re\_compile\_pattern ile iklendirilmesi20  
 tamponun iklendirilmesi .....19  
 used alanı  
 re\_compile\_pattern ile iklendirilmesi19

**T**

translate iklendirmesi .....19

**V**

VEYA işleci .....11  
 VEYA işleci ve ^ .....15

## Notlar

- a) Belge içinde dipnotlar ve dış bağlantılar varsa, bunlarla ilgili bilgiler buldukları sayfanın sonunda dipnot olarak verilmeyip, hepsi toplu olarak burada listelenmiş olacaktır.
- b) Konsol görüntüsünü temsil eden sarı zeminli alanlarda metin genişliğine sığmayan satırların sığmayan kısmı `↵` karakteri kullanılarak bir alt satıra indirilmiştir. Sarı zeminli alanlarda `↵` karakteri ile başlayan satırlar bir önceki satırın devamı olarak ele alınmalıdır.
- (1) Bazı durumlarda, özel karakterleri sıradan yapmak için onları doğrudan doğruya öncelemek zorunda değilsiniz. Örneğin, bir *liste* (sayfa: 11) içerisinde pek çok karakter özel anlamını kaybeder. Ek olarak, `'RE_CONTEXT_INVALID_OPS` ve `RE_CONTEXT_INDEP_OPS` sözdizimi bitleri sıfırda (tarihsel sebeplerden dolayı), şayet işlemler özel bir anlam ihtiva etmiyorsa, özel karakterler sıradan kabul edilir; örneğin, `*` tarafından temsil edilen sıfır veya daha fazlası ile eşleştirme işleci, `*foo` düzenli ifadesinde kendisi ile eşleşir, çünkü kendinden önce gelen ve işlem yapabileceği bir ifade yoktur. Bu kötü bir örnektir, bununla birlikte, özel bir karakterin liste dışında sıradan olmasını istiyorsanız, ne olursa olsun öncelemek en iyi yoldur.
- (2) Regex bu nedenle, `^` karakterini liste içindeki ilk karakter olarak kabul etmez. Şayet herhangi bir nedenle bir eşleştirme listesinin ilk karakteri olarak `^` kullanırsanız, bu onu eşleşmeme listesi haline dönüştürür.
- (3) Bir karakter sınıfı tek bir karakterden oluşmadığından karakter sınıflarını bir aralığın başlangıcı ya da sonu olarak belirtemezsiniz.
- (4) Düzenli İfadeler, “şablon tamponu” isminden dolayı “şablonlar/kalıplar” olarak da adlandırılır.
- (5) Bütün büyük harfleri, karşılığı olan küçük harflere dönüştürmeye yarayacak bir tablo sadece bu amaç için çalışacaktır.

Bu dosya (regexinfo.pdf), belgenin XML biçiminin `TeXLive` ve `belgeler-xsl` paketlerindeki araçlar kullanılarak PDF biçimine dönüştürülmesiyle elde edilmiştir.

27 Şubat 2007