정규 표현식(正規表現式, regular expression, 간단히 regexp[1] 또는 regex, rational expression)[2][3] 또는 정규식(正規式)은 특정한 규칙을 가진 문자열의 집합을 표현하는 데 사용하는 형식 언어이다. 정규 표현식은 많은 텍스트 편집기와 프로그래밍 언어에서 문자열의 검색과 치환을 위해 지원하고 있으며, 특히 펄과 Tcl은 언어 자체에 강력한 정규 표현식을 구현하고 있다.

컴퓨터 과학의 정규 언어로부터 유래하였으나 구현체에 따라서 정규 언어보다 더 넓은 언어를 표현할 수 있는 경우도 있으며, 심지어 정규 표현식 자체의 문법도 여러 가지 존재하고 있다. 현재 많은 프로그래밍 언어, 텍스트 처리 프로그램, 고급 텍스트 편집기 등이 정규 표현식 기능을 제공한다. 일부는 펄, 자바스크립트, 루비, Tcl처럼 문법에 내장되어 있는 반면 닷넷 언어, 자바, 파이썬, POSIX C, C++ (C++11 이후)에서는 표준 라이브러리를 통해 제공한다. 그 밖의 대부분의 언어들은 별도의 라이브러리를 통해 정규 표현식을 제공한다.

정규 표현식은 검색 엔진, 워드 프로세서와 문서 편집기의 찾아 바꾸기 대화상자, 또 sed, AWK와 같은 문자 처리 유틸리티, 어휘 분석에 사용된다.

패턴

클레이니 스타를 변환한 것. (*s**는 0번 이상의 *s*를 뜻함) 정규 표현식이라는 문구는 일치하는 텍스트가 준수해야 하는 "패턴"을 표현하기 위해 특정한 표준의 텍스트 신택스를 의미하기 위해 사용된다. 정규 표현식의 각 문자(즉, 패턴을 기술하는 문자열 안의 각 문자)는 메타문자(특별한 의미로)로 이해되거나 정규 문자('문자 그대로', 즉 '리터럴'의 의미로)로 이해된다. 이를테면 정규식 a. *a*는 단지 'a'와 일치하는 리터럴 문자이며 .는 새 줄을 제외한 모든 문자와 일치시키는 메타 문자이다. 그러므로 이 정규식은 이를테면 'a', 'ax', 'a0'과 일치시킬 수 있다. 더불어, 메타문자와 리터럴 문자는 주어진 패턴의 텍스트를 식별하기 위해 사용할 수 있으며, 또 수많은 인스턴스를 처리하기 위해 사용할 수도 있다. 패턴 일치는 정확히 동일한 일치에서부터 매우 포괄적인 유사 일치(메타문자에 의해 제어)에 이르기까지 다양하다. 이를테면 .는 매우 포괄적인 패턴이며, [a-z]('a'부터 'z'까지의 모든 문자 일치)는 덜 포괄적이며 a는 정확한 패턴(단지 'a'만 일치)이다. 메타문자 문법은 다양한 입력 데이터의 텍스트 처리의 자동화를 지시하는 정확하고 유연한 방법을 통해 표준 ASCII 자판을 사용하여 입력하기 쉬운 형태로, 미리 기술된 대상을 표현하기 위해 설계되었다.

이러한 신택스를 가지는 매우 단순한 정규 표현식의 경우는 문서 편집기에서 2가지 방식으로 발음되는 동일한 단어를 위치시키는 것을 들 수 있으며, 이를테면 seriali[sz]e는 "serialise"와 "serialize"를 모두 일치시킨다. 와일드카드 또한 이를 성취할 수 있으나 와일드카드가 패턴화할 수 있는 것으로 국한된다. (메타문자가 더 적으며 더 단순한 언어 기반임)

와일드카드 문자를 사용하는 경우는 파일 목록에서 비슷한 이름을 글로브 처리하는 것인 반면, 정규 표현식은 일반적으로 텍스트 문자열과 패턴 일치가 되는 애플리케이션에 보통 사용된다. 이를테면 ^[\t]+|[\t]+$는 줄 끝이나 줄 맨앞에 여분의 공백을 일치시킨다. 어떠한 숫자라도 일치시키기 위해 쓰이는 진보한 정규 표현식은 [+−]?(\d+(\.\d+)?|\.\d+)([eE][+−]?\d+)?이다. 더 많은 예를 보려면 예 문단을 참고할 것.

정규 표현식은 스티븐 클레이니가 정규 집합(regular set)이라는 자신의 수학적 개념을 이용하여 정규 언어를 기술한 1956년이 기원이다.[4] 형식 언어와 관련된 오토마타 이론의 하위 분야이자 이론 전산학에서 발생하였다. 패턴 일치의 초기 구현체들에는 SNOBOL 언어가 있으며, 실제로 정규식을 사용한 것은 아니지만 대신 독자적인 패턴 일치 구성체를 이용하였다.

여러 형태의 정규 표현식들이 1970년대에 벨 연구소에서 유닉스[5] 프로그램에 사용되었는데, 이를테면 Vi, lex, sed, awk, expr, Emacs를 포함한다. 그 뒤로 정규 표현식은 1992년 POSIX.2에 표준화된 초기 형태들을 이용하여 다양한 프로그램에 채택되었다.

1980년대에 더 복잡한 정규 표현식이 펄에 등장하였으며 이는 1986년 헨리 스펜서가 작성한 정규 표현식 라이브러리에서 기인하며 나중에 이 작성자는 Tcl을 위한 고급 정규 표현식의 구현체를 작성하기도 했다. 펄 6

에서는 펄의 정규 표현식 통합을 개선함과 동시에 파싱 표현 문법(parsing expression grammer)의 정의를 허용하기 위해 범위와 기능을 증가시키려는 노력이 있었다.

1997년을 기점으로 필립 하젤은 펄의 정규 기능을 모방하려는 시도 속에 PCRE를 개발하였으며 PHP와 아파치 HTTP 서버를 포함한 여러 현대적인 도구들에 사용된다.

오늘날 정규 표현식은 프로그래밍 언어, 문자 처리 프로그램(특히 lexer), 고급 문서 편집기 등의 프로그램에서 널리 지원된다. 정규 표현식 지원은 자바와 파이썬을 포함한 여러 프로그래밍 언어의 표준 라이브러리의 일부가 되었으며, 펄과 ECMA스크립트에서는 기본 문법으로 통합되었다.

기본 개념

주로 **패턴**(pattern)으로 부르는 정규 표현식은 특정 목적을 위해 필요한 문자열 집합을 지정하기 위해 쓰이는 식이다. 문자열의 유한 집합을 지정하는 단순한 방법은 문자열의 요소나 멤버를 나열하는 것이다. 그러나 문자열의 원하는 집합을 지정하기 위해 사용할 수 있는 더 간결한 방법들이 있다. 이를테면 3개의 문자열 "Handel", "Händel", "Haendel"을 포함하는 집합은 패턴 H(ä|ae?)ndel으로 지정이 가능하며 이러한 패턴은 3개의 문자열을 각각 **일치**(match)시킨다고 이야기한다.

불리언 "또는"
수직선은 여러 항목 중 선택을 하기 위해 구분한다. 이를테면 gray|grey는 "gray" 또는 "grey"와 일치한다.

그룹 묶기
괄호를 사용하면 연산자의 범위와 우선권을 정의할 수 있다. 이를테면 gray|grey와 gr(a|e)y는 "gray"나 "grey" 집합을 둘 다 기술하는 동일 패턴이다.

양의 지정

?	물음표는 0번 또는 1차례까지의 발생을 의미한다. 이를테면 colou?r는 "color"와 "colour"를 둘 다 일치시킨다.
*	별표는 0번 이상의 발생을 의미한다. 이를테면 ab*c는 "ac", "abc", "abbc", "abbbc" 등을 일치시킨다.
+	덧셈 기호는 1번 이상의 발생을 의미한다. 이를테면 ab+c는 "abc", "abbc", "abbbc" 등을 일치시키지만 "ac"는 일치시키지 않는다.
{n}[6]	정확히 n 번만큼 일치시킨다.
{min,}[6]	"min"번 이상만큼 일치시킨다.
{min,max}[6]	적어도 "min"번만큼 일치시키지만 "max"번을 초과하여 일치시키지는 않는다.

문법

정규 표현식의 패턴은 대상 문자열과 일치시킨다. 이 패턴은 일련의 원자로 구성된다. 하나의 원자는 정규 표현식 패턴 안의 하나의 점이며, 대상 문자열과 일치시키기 위해 존재한다. 가장 단순한 형태의 원자는 리터럴(literal)이지만 원자를 일치시키기 위해 패턴을 묶을 때에는 메타문자로서 ( )를 사용해야 한다. 메타문자는 원자 외에도 얼마나 많은 원자가 있는지를 가리키는 수량자(quantifier), 선택적인 대안을 가리키는 논리적 OR 문자, 그리고 원자의 존재를 부정하는 NOT 문자, 그리고 전에 일치된 원자를 참조할 수 있게 하는 역참조(backreference)를 만드는 것을 도와준다.

구분 문자

프로그래밍 언어에서 정규 표현식을 입력할 때, 통상적인 문자열 리터럴로 표현할 수 있으므로 일반적으로 인용 부호로 처리된다. 이는 이를테면 C, 자바, 파이썬에서도 동일한데, 정규표현식 re는 "re"로 입력된다. 그러나 구분 문자로 슬래시를 사용하는 경우도 있는데 정규 표현식 re에 대해서 /re/를 사용하는 식이다. 이는 ed라는 편집기에서 기원하며, /는 검색을 위한 편집기 명령어이고, /re/라는 식은 특정한 범위의 줄들을 지정하는데 사용할 수 있다. 비슷한 방식이 sed에서 사용되는데, 검색 후 치환은 s/re/replacement/를 사용하며 패턴은 /re1/,/re2/처럼 특정한 범위의 줄을 지정하여 쉼표(,)로 결합할 수 있다. 이러한 표기법은 특히 펄에서도 사용된다.

표준

POSIX 기본 및 확장 표준 문법

문자 클래스, "["와 "]" 사이에 포함된 문자 집합 외부에서는 12개의 문자가, 내부에서는 오직 4개의 문자("\", "^", "-", "]", 자바와 닷넷은 "["를 포함)만 특수문자를 의미한다.[7] 아래는 POSIX 기본 및 확장 표준의 문법이다.

메타문자	기능	설명
.	문자	1개의 문자와 일치한다. 단일행 모드에서는 새줄 문자를 제외한다.
[ ]	문자 클래스	"["과 "]" 사이의 문자 중 하나를 선택한다. "¦"를 여러 개 쓴 것과 같은 의미이다. 예를 들면 [abc]d는 ad, bd, cd를 뜻한다. 또한, "-" 기호와 함께 쓰면 범위를 지정할 수 있다. "[a-z]"는 a부터 z까지 중 하나, "[1-9]"는 1부터 9까지 중의 하나를 의미한다.
[^]	부정	문자 클래스 안의 문자를 제외한 나머지를 선택한다. 예를 들면 [^abc]d는 ad, bd, cd는 포함하지 않고 ed, fd 등을 포함한다. [^a-z]는 알파벳 소문자로 시작하지 않는 모든 문자를 의미한다.
^	처음	문자열이나 행의 처음을 의미한다.
$	끝	문자열이나 행의 끝을 의미한다.
( )	하위식	여러 식을 하나로 묶을 수 있다. "abc¦adc"와 "a(b¦d)c"는 같은 의미를 가진다.
₩n	일치하는 n번째 패턴	일치하는 패턴들 중 n번째를 선택하며, 여기에서 n은 1에서 9 중 하나가 올 수 있다.
*	0회 이상	0개 이상의 문자를 포함한다. "a*b"는 "b", "ab", "aab", "aaab"를 포함한다.
{m, n}	m회 이상 n회 이하	"a{1,3}b"는 "ab", "aab", "aaab"를 포함하지만, "b"나 "aaaab"는 포함하지 않는다.

많은 프로그래밍 언어에서는 이를 확장한 문법을 가지고 있다. 이 중 POSIX에서 사용되는 확장 문법은 POSIX 확장 문법을 참고할 것. 그 외의 환경에 대해서는 문자 클래스 단락을 참고할 것.

POSIX 확장 문법

메타문자	기능	설명
?	0 또는 1회	"a?b"는 "b", "ab"를 포함한다.
+	1회 이상	"a+b"는 "ab", "aab", "aaab"를 포함하지만 "b"는 포함하지 않는다.
¦	선택	여러 식 중에서 하나를 선택한다. 예를 들어, "abc¦adc"는 abc와 adc 문자열을 모두 포함한다.

문자 클래스

문자 클래스는 문자열 일치 다음으로 가장 기본적인 정규 표현식 개념이다. 이는 하나의 작은 일련의 문자열들을 더 큰 집합의 문자열들과 일치시키도록 한다. 이를테면, [A-Z]는 알파벳을 대표하며 \d는 임의의 숫자를 의미할 수 있다. 문자 클래스는 POSIX 수준에 적용한다.

[a-Z]와 같은 특정 범위의 문자들을 지정할 때 컴퓨터의 로캘 설정들은 문자 인코딩의 수치적 나열에 따라 내용을 결정한다. 그러한 나열에 따라 수치들을 저장할 수 있으며 그 순서는 *abc...zABC...Z, aAbBcC...zZ*와 같이 될 수 있다. 그러므로 POSIX 표준은 문자 클래스를 정의하며 이는 설치된 정규 표현식 처리기가 인지한다. 이러한 정의들은 다음의 표를 따른다:

POSIX	비표준	펄/Tcl	Vim	ASCII	설명
[:alnum:]				[A-Za-z0-9]	영숫자
	[:word:]	₩w	₩w	[A-Za-z0-9_]	영숫자 + "_"
		₩W	₩W	[^A-Za-z0-9_]	낱말이 아닌 문자
[:alpha:]			₩a	[A-Za-z]	알파벳 문자
[:blank:]			₩s	[₩t]	공백과 탭
		₩b	₩< ₩>	(?<=₩W)(?=₩w)|(?<=₩w)(?=₩W)	낱말 경계
[:cntrl:]				[₩x00-₩x1F₩x7F]	제어 문자
[:digit:]		₩d	₩d	[0-9]	숫자
		₩D	₩D	[^0-9]	숫자가 아닌 문자
[:graph:]				[₩x21-₩x7E]	보이는 문자
[:lower:]			₩l	[a-z]	소문자
[:print:]			₩p	[₩x20-₩x7E]	보이는 문자 및 공백 문자
[:punct:]				[][!"#$%&'()*+,./:;<=>?@₩^_`{|}~-]	구두점
[:space:]		₩s	₩_s (단순히 줄 끝에 추가)	[₩t₩r₩n₩v₩f]	공백 문자
		₩S		[^ ₩t₩r₩n₩v₩f]	공백이 아닌 모든 문자
[:upper:]			₩u	[A-Z]	대문자
[:xdigit:]			₩x	[A-Fa-f0-9]	16진수

펄과 PCRE

표현력과 (상대적으로) 양호한 가독성으로 인해 자바, 자바스크립트, 파이썬, 루비, 마이크로소프트의 닷넷 프레임워크, XML 스키마 등의 여러 수많은 유틸리티들과 프로그래밍 언어들은 펄의 정규식과 비슷한 문법을 채택하고 있다. Boost와 PHP와 같은 일부 언어들과 도구들은 여러 방식의 정규식을 지원한다. 펄 파생 정규식 구현체들은 완전히 동일한 것은 아니며 일반적으로 1994년에 출시된 펄 5.0의 기능의 하위 집합을 구현하고 있다. 펄은 종종 다른 언어에서 발견되는 기능들을 도입하고 있는데, 이를테면 펄 5.10은 PCRE와 파이썬에서 개발되는 문법 확장들을 구현한다.[8]

이용

정규 표현식은 다양한 텍스트 처리 작업, 더 일반적으로 문자열 처리에 유용하며, 여기에서 데이터는 반드시 텍스트일 필요는 없다. 일반적으로 데이터 검증, 데이터 스크래핑(일반적으로 웹 스크래핑), 데이터 랭글링, 단순 파싱, 문법 강조 시스템 개발 등의 작업에 응용된다.

정규 표현식이 인터넷 검색 엔진에 유용할 수 있으나 전체 데이터베이스를 거쳐 이것들을 처리하기에는 정규 표현식의 복잡도와 설계에 따라 컴퓨터 자원을 과도하게 소비할 수 있다. 대개의 경우 시스템 관리자들이 정규 표현식 기반 쿼리를 내부적으로 수행할 수 있지만 대부분의 검색 엔진들은 대중에게 정규 표현식 지원을 제공하지 않는다.

예

특정 문법 규칙은 사용 중인 특정 구현체, 프로그래밍 언어, 라이브러리에 따라 다양하다. 또한 정규 표현식 구현체들의 기능은 소프트웨어 버전 간에도 다를 수 있다.

정규 표현식들은 예제 없이 설명과 이해를 동시에 하는 것은 어려울 수 있다. 아래에는 정규 표현식의 속성 중 일부의 기본적인 설명을 제시한다.

아래의 항목들은 예제에 사용된다.[9]

메타문자 ;; 메타문자들의 열은 표현할 정규식을 지정한다. =~ m// ;; 펄에서 문자열을 '일치'시키려는 동작을 지정한다. =~ s/// ;; 펄에서 문자열을 '대체'시키려는 동작을 지정한다.

.	일련의 문자들 중 줄을 제외한 모든 어떠한 문자열과도 일치한다.	if ($string1 =~ m/...../) { print "$string1 has length >= 5₩n"; } 출력: Hello World has length >= 5
( )	일련의 패턴 요소들을 하나의 요소로 묶는다. 괄호 안의 패턴을 일치시킬 때 $1, $2, ... 중 하나를 사용할 수 있다.	$string1 = "Hello World₩n"; if ($string1 =~ m/(H..).(o..)/) { print "We matched '$1' and '$2'₩n";} 출력:We matched 'Hel' and 'o W'
+	1번 이상 발생하는 패턴과 일치시킨다.	$string1 = "Hello World₩n"; if ($string1 =~ m/l+/) { print "There are one or more consecutive letter ₩"l₩"'s in $string1₩n";} 출력:There are one or more consecutive letter "l"'s in Hello World
?	0~1번 발생하는 패턴과 일치시킨다.	$string1 = "Hello World₩n";if ($string1 =~ m/H.?e/) { print "There is an 'H' and a 'e' separated by "; print "0-1 characters (Ex: He Hoe)₩n";} 출력:There is an 'H' and a 'e' separated by 0-1 characters (Ex: He Hoe)
?		$string1 = "Hello World₩n"; if ($string1 =~ m/(l.+?o)/) { print "The non-greedy match with 'l' followed by one or₩n"; print "more characters is 'llo' rather than 'llo Wo'.₩n";} 출력: The non-greedy match with 'l' followed by one ormore characters is 'llo' rather than 'llo Wo'.
*	0번 이상 발생하는 패턴과 일치시킨다.	$string1 = "Hello World₩n";if ($string1 =~ m/el*o/) { print "There is an 'e' followed by zero to many "; print "'l' followed by 'o' (eo, elo, ello, elllo)₩n";} 출력: There is an 'e' followed by zero to many 'l' followed by 'o' (eo, elo, ello, elllo)
{M,N}	최소 M번, 최대 N번 발생되는 패턴과 일치시킨다.	$string1 = "Hello World₩n";if ($string1 =~ m/l{1,2}/) { print "There exists a substring with at least 1 "; print "and at most 2 l's in $string1₩n";} 출력: There exists a substring with at least 1 and at most 2 l's in Hello World
[...]	가능한 문자열의 집합과 일치시킨다.	$string1 = "Hello World₩n";if ($string1 =~ m/[aeiou]+/) { print "$string1 contains one or more vowels.₩n";} 출력: Hello World contains one or more vowels.
|	가능성 있는 항목들을 구별하여 선택한다.	$string1 = "Hello World₩n";if ($string1 =~ m/(Hello|Hi|Pogo)/) { print "At least one of Hello, Hi, or Pogo is "; print "contained in $string1.₩n";} 출력: At least one of Hello, Hi, or Pogo is contained in Hello World.
₩b		$string1 = "Hello World₩n";if ($string1 =~ m/llo₩b/) { print "There is a word that ends with 'llo'₩n";} 출력: There is a word that ends with 'llo'
₩w	"_"를 포함한 영숫자를 일치시킨다.	$string1 = "Hello World₩n";if ($string1 =~ m/₩w/) { print "There is at least one alphanumeric "; print "character in $string1 (A-Z, a-z, 0-9, _)₩n";} 출력: There is at least one alphanumeric character in Hello World (A-Z, a-z, 0-9, _)
₩W	"_"를 제외하여 영숫자가	$string1 = "Hello World₩n";if ($string1 =~ m/₩W/) { print "The space

	아닌 문자열들과 일치시킨다.	between Hello and "; print "World is not alphanumeric₩n";} 출력: The space between Hello and World is not alphanumeric
₩s	공백 문자와 일치시킨다.	$string1 = "Hello World₩n";if ($string1 =~ m/₩s.*₩s/) { print "There are TWO whitespace characters, which may"; print " be separated by other characters, in $string1";} 출력: There are TWO whitespace characters, which may be separated by other characters, in Hello World
₩S	공백을 제외한 어떠한 것이든 일치시킨다.	$string1 = "Hello World₩n";if ($string1 =~ m/₩S.*₩S/) { print "There are TWO non-whitespace characters, which"; print " may be separated by other characters, in $string1";} 출력: There are TWO non-whitespace characters, which may be separated by other characters, in Hello World
₩d	숫자를 일치시킨다.	$string1 = "99 bottles of beer on the wall.";if ($string1 =~ m/(₩d+)/) { print "$1 is the first number in '$string1'₩n";} 출력: 99 is the first number in '99 bottles of beer on the wall.'
₩D	숫자가 아닌 항목을 일치시킨다.	$string1 = "Hello World₩n";if ($string1 =~ m/₩D/) { print "There is at least one character in $string1"; print " that is not a digit.₩n";} 출력: There is at least one character in Hello World that is not a digit.
^	줄이나 문자열의 시작점과 일치시킨다.	$string1 = "Hello World₩n";if ($string1 =~ m/^He/) { print "$string1 starts with the characters 'He'₩n";} 출력: Hello World starts with the characters 'He'
$	줄이나 문자열의 끝과 일치시킨다.	$string1 = "Hello World₩n";if ($string1 =~ m/rld$/) { print "$string1 is a line or string "; print "that ends with 'rld'₩n";} 출력: Hello World is a line or string that ends with 'rld'
₩A	문자열의 시작점과 일치시킨다. (내부 줄이 아닌)	$string1 = "Hello₩nWorld₩n"; if ($string1 =~ m/₩AH/) { print "$string1 is a string "; print "that starts with 'H'₩n";} 출력: HelloWorld is a string that starts with 'H'
₩z	문자열의 끝과 일치시킨다. (내부 줄이 아닌)[12]	$string1 = "Hello₩nWorld₩n";if ($string1 =~ m/d₩n₩z/) { print "$string1 is a string "; print "that ends with 'd₩₩n'₩n";} 출력: HelloWorld is a string that ends with 'd₩n'
[^...]	괄호 안의 항목을 제외한 모든 문자열과 일치시킨다.	$string1 = "Hello World₩n";if ($string1 =~ m/[^abc]/) { print "$string1 contains a character other than "; print "a, b, and c₩n";} 출력: Hello World contains a character other than a, b, and c