정규 표현식

차리서 <reeseo@konkuk.ac.kr>

건국대학교 공과대학 컴퓨터공학부

학기 전체 일정

주	월	화	수	목	금	토	일	실습	강의
1	8月 28	29	30	31	1	2	3	수강 정정, 팀 결성	과목 오리엔테이션, 팀 결성 안내
2	9月 4	5	6	7	8	9	10	팀 결성 완료, 주제 선정 시작	주제 선정 안내, 기획서 안내
3	11	12	13	14	15	16	17	**주제 선정 완료, 기획 시작**	**기획서 안내**
4	18	19	20	21	22	23	24	기획서 작성	설계 문서 안내
5	25	26	27	28	29	30	1	설계 문서 작성	(설계 문서 문답)
6	10月 2	3	4	5	6	7	8	설계 문서 작성	요구사항 분석 안내
7	9	10	11	12	13	14	15	요구사항 분석 및 재설계	(요구사항 분석, 재설계 문답)
8	16	17	18	19	20	21	22	요구사항 분석 및 재설계	구현 및 검사 안내, 중간 발표 안내
9	23	24	25	26	27	28	29	구현 및 검사	(구현 및 검사 문답)
10	30	31	1	2	3	4	5	구현 및 검사	(구현 및 검사 문답)
11	11月 6	7	8	9	10	11	12	중간 발표	—
12	13	14	15	16	17	18	19	요구사항 분석, 재설계/구현	(재설계/구현 문답)
13	20	21	22	23	24	25	26	요구사항 분석, 재설계/구현	(재설계/구현 문답)
14	27	28	29	30	1	2	3	요구사항 분석, 재설계/구현, 검사	(재설계/구현 문답)
15	12月 4	5	6	7	8	9	10	요구사항 분석, 재설계/구현, 검사	기말 발표 안내
16	11	12	13	14	15	16	17	기말 발표	—

이 수업에서의 정규 표현식

기획서/요구분석서와 설계서에서:

- *사용하지* ***않아도*** *됨*
- **사용할 경우, 자연어와** ***병기해도 되고*** **자연어를 대체해도 됨**
  - 자연어와 병기할 경우 (권장): 자연어는 반드시 '맞아야'하고, 정규 표현식은 틀려도 됨
  - 자연어를 대체할 경우: 틀리면 안 됨!

구현에서 (라이브러리를):

- 사용하지 않아도 됨
- 사용할 경우, 맞으면 상당히 편하지만 틀리면 구현 불가
- 사용하기 곤란한 경우도 있음 (윈도 플랫폼의 Visual Studio 기반 C 언어인 경우 등)

촘스키 위계Chomsky hierarchy

언어학자 노엄 촘스키의 형식주의 언어론에 의거한 언어 분류 방식

유형	문법grammar	언어language, set	인식하는 자동 기계
Type-0	Unrestricted Grammar 무제약 문법	Recursively Enumerable set 재귀 열거 언어	Turing machine 튜링 머신
Type-1	Context-Sensitive Grammar 문맥 의존 문법	Context-Sensitive Language 문맥 의존 언어	Linear-bounded non-deterministic Turing machine 선형경계 비결정적 튜링 머신
Type-2	Context-Free Grammar 문맥 자유 문법	Context-Free Language 문맥 자유 언어	Non-deterministic pushdown automaton 비결정적 푸시다운 오토마타
Type-3	**Regular Grammar** **정규 문법**	**Regular Language** **정규 언어**	**Finite state automaton** **유한 상태 기계**

- ‘언어’란 (그 언어의 ‘문법’에 부합하는) 문자열들의 집합
- n유형 언어는 n-1유형 언어의 진부분집합: $L_{\text{Regular}} \subset L_{\text{CF}} \subset L_{\text{CS}} \subset L_{\text{RE}}$
- ‘인식하는 자동 기계’에 관해서는 계산 이론[1] 과목에서 각 유형별로 자세히 배우게 됨

[1] 건국대 컴퓨터공학부 기준으로, 가을 학기에 개설되는 BBAB53292 “컴퓨테이션 이론”

(결정적Deterministic) 유한 상태 기계

결정적 vs. 비결정적non-deterministic 유한 상태 기계

- 수학적인 정의와 동작 방식, 도식적 표현은 서로 다름
- 계산 (판정) 능력은 서로 완전히 동일 → 이 강의에서는 결정적 유한 상태 기계만 설명

결정적 유한 상태 기계의 정의: $\langle Q, \Sigma, \delta, q_0, F \rangle$ 의 5-튜플

- 상태들의 유한 집합 Q
- 입력 문자열에 허용되는 모든 문자들의 유한 집합 Σ (이 집합을 흔히 "알파벳"이라고도 부름)
- 상태 전이 함수 $\delta : Q \times \Sigma \to Q$
  - (현재) 상태 하나와 (입력) 문자 하나를 인자로 넣으면 (다음) 상태를 리턴하는 함수
- 초기 (즉, 시작) 상태 $q_0 \in Q$
- 종료 상태(들)의 집합 $F \subseteq Q$

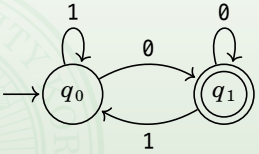
초기 상태 q_0 에서 시작해서, 주어진 입력 문자열의 각 문자들을 순서대로 소모하면서, 그때마다 매번 (현재 상태와 소모된 문자에 따라) 다음 상태로 계속 이동해서, 입력 문자열을 모두 소모한 후 도착한 상태가 종료 상태 중 하나면 (F의 원소면) accept, 아니면 non-accept.

예시: 이진수 짝수 판정 (정의, 도식화, 정규표현식)

알파벳 $\Sigma = \{0, 1\}$일 때, **'이진수로 해석했을 때 (선행 0들도 허용하면서) 짝수인' (즉, 0으로 끝나는) 문자열**인지 판정해주는 기계:

$\langle \{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\} \rangle$ 단,

$$\begin{aligned} \delta(q_0, 0) &= q_1 \\ \delta(q_0, 1) &= q_0 \\ \delta(q_1, 0) &= q_1 \\ \delta(q_1, 1) &= q_0 \end{aligned}$$



도식화:

- 허공에서 출발하는 화살표는 딱 하나만 있어야 하며, 이 화살표가 가리키는 상태가 초기 상태
- 종료 상태(들)는 이중 테두리로 표시
- 각 상태마다, 그 상태에서 출발하는 모든 화살표들의 표찰을 전부 모으면 정확히 알파벳(Σ)이어야 함

더 간단히 표현할 수 없을까? → 정규표현식

```
[01]*0
```

정규 표현식 Regular Expression

정규 표현식 (혹은 “정규식”): 정규 문법을 나타내는 표현법

- 정규 표현식으로 나타낼 수 있는 문법은 반드시 일종의 정규 문법
- 몇 가지 문자들은 특별한 의미로 약속: (개행 등의) 특수 문자, 택일, 그룹, 반복 등을 나타냄
- 대체로 “RegExp”, “Regex”, 혹은 “RE”라고 표기된 것들(라이브러리 등)이 바로 정규 표현식

서로 다른 정규식 문법 체계가 다수 존재

- 각 문자를 escape했을 때 무엇을 의미하고 안 했을 때 무엇을 의미하는가?
- 특별한 의미의 문자들이 (기본 공통 문자들 외에) 무엇 무엇이 더 있는가?
- 심지어 일부는 ‘기억’을 위한 추가 기능도 제공: 이러면 더이상 (엄밀한 의미에서) 정규식이 아님
- POSIX 기본 정규식, POSIX 확장 정규식, PCRE (Perl 언어 호환 정규식)[2], Vim 정규 표현식 등
- 이 수업에서는 각 확장별 추가 기능들은 거의 빼고 공통 기본 기능 위주로 안내

[2] Perl 언어에서 유래된 체계지만, 현재는 그 유래/이름과는 달리 수많은 언어/도구에서 기본으로 사용하거나 선택 가능.

대응match 개념

어떤 문자열 S가 어떤 정규 표현식 R이 나타내는 문자열들 중 하나라면, 이를 다음과 같이 말 함:

"정규 표현식 R이 문자열 S에 **대응한다**."
"The regex R **matches** the string S."

예: 정규 표현식 "`[01]*0`"은 문자열 "`0`", "`0010`", "`1011100`" 등에 대응함.

본인들이 의도한 '올바른' 입력 문자열이 S_1, S_2, ..., S_n 뿐일 때, 이를 정규 표현식 R로 나타낸다면:

- **필수: R은 S_1, S_2, ..., S_n 모두에 예외 없이 대응해야 함!**
- 권장: R은 S_1, S_2, ..., S_n **에만** 대응하는 게 좋음. 만일 이들 외에 대응하는 문자열이:
  - 더 있다면, 그 나머지를 걸러내기 위한 의미 규칙을 (자연어나 수식 등으로) 추가로 명시해야 함
  - 더 없다면, 이 정규 표현식이 문법 규칙이자 동시에 의미 규칙임 (즉, 별도의 의미 규칙 불필요)
- 때로는 '문법'과 '의미'를 엄격히 구분하기 위해 의도적으로, 문법(정규 표현식)으로는 일단 허용(대응)한 후 2차로 의미 규칙을 통해 거르는 방식도 사용 (예: "`3 / 0`")

리터럴 문자literal characters

정규 표현식도 (특별한 의미의 기호가 섞여있을 뿐) 결국은 문자열!

- 리터럴이란 원래 “문자 그대로, 쓰여있는 그대로”라는 의미
- 정규 표현식 속의 (특별한 의미가 없는) 일반 문자는 실제 그 자리의 그 문자에 대응
- 예: 정규 표현식 “abc”는 문자 그대로 “abc”라는 문자열에 대응

화면 상에 표시하기 곤란한 문자들 중에는 특별한 방식으로 입력/표시하는 문자들이 다수 존재:[3)]

`\t` 탭 문자

`\n` 개행 문자[4)]

주의: 나중에 설명할 ‘리터럴 문자 → 메타 문자’ 변환과는 다름. 위 목록은 메타 문자가 아니라 역슬래시가 붙어있는 상태 자체가 리터럴임.

평범한 공백 (스페이스 바) 문자는 화면 상에 빈 칸으로 표시 가능하지만, 시각적으로 모호한 경우 “␣” 기호 이용.

[3)] 이 외에도 상당히 많지만 수업에 (특히 기획서에) 필요한 건 이 둘이면 충분함. 또한, 이들 두 문자는 키입력된 문자열의 규칙보다는 주로 텍스트 파일의 규칙을 나타낼 때 필요.

[4)] 실제 개행은 LF(line feed)와 CR(carriage return)의 조합인데, 조합 방식이 플랫폼마다 다름. 이 수업에서는 그냥 본인들 팀이 사용하는 플랫폼의 개행을 \n 문자 하나로 나타낸다고 가정.

메타 문자metacharacters

정규 표현식은 (결국은 문자열이지만) 특별한 의미의 기호가 섞여있음!

기호	의미	기타
`\`	escaping	바로 다음 페이지 참고
`^`	(개행으로 나뉜) 문장의 첫 지점	“기타” 절 참고
`$`	(개행으로 나뉜) 문장의 끝 지점	
`.`	임의의 한 문자에 대응	“택일” 절 참고
`[ ]`	문자 택일 나열을 만드는 구분자	
`^`	문자 택일 나열 속에서 여집합 표시	
`-`	문자 택일 나열 속에서 범위 표시	
`¦`	문자열, 그룹 택일 나열을 만드는 구분자	
`( )`	그룹을 만드는 구분자	“반복” 절 참고
`?`	0 개 혹은 1 개	
`*`	0 개 이상	
`+`	1 개 이상	
`{ , }`	~ 개 이상 ~ 개 이하	

Escaping: 메타 문자 → 리터럴 문자

Q: 온점(.)이 메타 문자면, 소수점 기호 자체는 정규 표현식 속에서 어떻게 표시하나요?

A: 메타 문자를 리터럴로 바꾸는 방법: 메타 문자 앞에 역슬래시 붙임

- `\\` `\^` `\$` `\.` `\[` `\]` `\-` `\|` `\(` `\)` `\?` `\*` `\+` `\{` `\}`
- 예: "`.`"은 아무 문자에나 대응하는 메타 문자지만, "`\.`"은 문자 그대로 온점(마침표) 문자에 대응
- "`,`"은 원래 "`{ }`" 안에서만 메타 문자: 밖에서는 역슬래시 없이 그냥 써도 리터럴
- "`-`"은 원래 "`[ ]`" 안에서(그것도 일부 경우에)만 메타 문자: 밖에서는 그냥 써도 리터럴

주의: 프로그래밍 언어/도구마다 세부 사항이 다름

- 일부 메타 문자는 위와 반대로 역슬래시가 붙어야 메타 문자, 안 붙으면 리터럴
- 거의 달라지지 않는 공통 문자들: `\` `^` `$` `.` `[` `]` `-` `?` `*`
- 언어/도구마다 자주 달라지는 문자들: `|` `(` `)` `+` `{` `}`
- 이 과목에서 사용할 경우 이 슬라이드가 기준이되, 각 팀별 상황에 따라 **제대로 명시하고** 변경 가능
- 예: 산술식 입력 때문에 입력 문자열 중에 괄호와 +가 리터럴로 자주 등장하는 팀:
  → "이 기획서에 사용된 정규 표현식에서 (,), +은 리터럴이고 \(, \), \+은 메타 문자입니다."

정해진 문자들 중 택일: 메타 문자 [ ]

`[ ]` 안쪽에 나열된 여러 **문자**들 중 (대응시킬 수 있는) 딱 한 문자를 의미하는 기호

- 사각 괄호 안에는 반드시 (뒤에서 소개할 딱 두 개의 메타 문자 외에는) 리터럴 문자만 들어가야 함[5)]
- 예: 정규식 “`[ABF]`”는 문자 “A”, “B”, “F”에(만) 대응
- 예: 정규식 “`[0123456789]`”는 아무 10진법 숫자 하나에(만) 대응
- 예: 정규식 “`[0123456789abcdefABCDEF]`”는 아무 16진법 숫자 하나에(만) 대응
- 예: 정규식 “`un[hpt]ac[kt]`”는 문자열 “unhack”, “unhact”, “unpack”, “unpact”, “untack”, “untact”에(만) 대응
- 예: 정규식 “`Get[␣\t]off!`”는 문자열 “`Get␣off!`”, “`Get⟶off!`”에(만) 대응

[5)] `\t`와 `\n`, 그리고 (스페이스 바로 입력하는) 빈 칸은 리터럴 문자임을 상기할 것.

정해진 문자들 중 택일: 메타 문자 [ ] 속의 메타 문자 "-" (범위)

`[ ]` 내부에서 **두 리터럴 문자 사이**에 사용된 `-` 문자는 범위를 뜻하는 메타 문자

- 안쪽에 "`f-t`"라는 형태가 보일 때마다, 이것을 일단 "`fghijklmnopqrst`"로 치환해놓는 방식
- 순차적으로 나열할 수 있는 문자라야 함. (대체로) 문자 코드 값 기준으로 동작
- "`f-t`" 형태가 여러 개 등장해도 됨
- 내부 치환이 다 끝나고 나면, 치환된 `[...]` 중에서 한 문자 선택
- 예: 정규식 "`[0-9]`"는 아무 10진법 숫자 하나
- 예: 정규식 "`[0-9a-fA-F]`"는 아무 16진법 숫자 하나
- 예: 정규식 "`20[01][0-9]`"는 2000부터 2019까지의 정수 중 하나

리터럴 문자 "`-`" 자체를 선택지 중 하나로 넣고 싶을 때: 대괄호 속의 맨 처음이나 맨 마지막에 위치

- 예: 정규식 "`[-\+][01]`"은[6] 문자열 "`-0`", "`-1`", "`+0`", "`+1`" 중 하나

[6] "`+`" 자체는 메타 문자기 때문에 역슬래시로 escape해서 리터럴로 바꿔야 함.

정해진 문자들 중 택일: 메타 문자 [ ] 속의 메타 문자 "^" (여집합)

`[ ]` 내부의 **제일 처음**에 사용된 `^` 문자는 여집합을 뜻하는 메타 문자

- 결국, 그냥 `^`까지 합쳐진 "`[^ ]`" 자체를 여집합을 뜻하는 한 덩어리의 괄호(메타문자)로 생각해도 됨
- '이 속에 들어있는 문자들만 빼고, 나머지 모든 글자'라는 의미
- 앞 페이지의 메타 문자 "`-`"와 동시에 사용할 수 있음: 해석 방법은 동일
- 예: 정규식 "`201[^3-5]`"는 2013, 2014, 2015만 제외한 나머지 2010년대 연도 **...가 아님!!**
  - 이 정규식은 "`2010`", "`2011`" 등등 뿐만 아니라 "`201a`", "`201b`", 심지어 "`201뷁`" 에도 대응됨!
  - 분명히 나머지 **모든** 문자라고 했음: 자주 실수를 유발하는 부분이니 주의할 것!
  - 그럼 이걸 어디에 쓰나요? → 뒤에 나올 '반복'과 합쳐서 텍스트 파일의 (예를 들어 탭 문자로 서로 구분된) 각 필드를 골라내는 일 등에 유용

아무 문자나 하나 택일: 메타 문자 .

(개행 문자를 제외한) 어떤 문자든 딱 한 개의 문자에 대응

- 예: 정규식 “`a.z`”는 문자열 “aaz”, “abz”, “acz”, ..., “azz”, 심지어 “a0z”, “a뷁z” 등에도 대응
- 예: 정규식 “고..”은
  - 성이 고 씨고 이름이 두 글자인 사람들의 전체 이름에 대응될 뿐만 아니라
  - (주의) 심지어 “고YJ”, “고␣5”, “고⟶␣”에도 대응됨
- 예: 정규식 “.....”은 길이가 5인 아무 문자열에나 다 대응

진짜 온점(마침표)을 표현하려면? → `\.` 으로 escape시켜야 함

정해진 문자열들 중 택일: 메타 문자 ¦

¦ 를 경계로 나뉜 문자열(선택지)들에 대응: 선택지 갯수는 2 개 이상, 각 선택지 길이는 0 문자 이상

예: 정규식 “abc¦de¦1234”는 문자열 “abc”, “de”, “1234”에(만) 대응

왼쪽 끝과 오른쪽 끝 경계를 표시할 때에는 괄호를 (반드시 짝지어서) 사용

예: 정규식 “**전공(필수¦선택)과목**”은 문자열 “전공필수과목”, “전공선택과목”에(만) 대응

각 선택지 문자열 속에는 다시 정규 표현식 메타 문자들이 사용될 수 있음

예: 정규식 “(197[4-9]¦19[89][0-9]¦20[01][0-9]¦202[0-3])**년**”은 1974년부터 올해까지

→ 4개의 선택지들 중 하나를 고르는 상황

예: 정규식 “(197[4-9]¦(19[89]¦20[01])[0-9]¦202[0-3])**년**”도 1974년부터 올해까지

→ 3개의 선택지들 중 하나를 고르는 상황

→ 2번째 선택지 속에 ‘2개의 선택지들 중 하나를 고르는 상황’이 들어있음

모든 선택지가 길이 1 짜리 문자열(즉, 문자 한 개)이라면 [ ]와 같은 효과

예: 정규식 “201(5¦6¦7¦8¦9)**년**”은 정규식 “201[5-9]**년**”과 동일

그룹 (반복 횟수의 적용 대상)

반복 횟수를 나타내는 메타 문자는 일종의 후위postfix 연산자: 적용 대상을 정확히 가리키는 규칙 필요

- 리터럴 문자는 그 자체가 그룹
- 메타 문자 중 `.` 문자는 그 자체가 그룹
- 메타 문자 중 `[`와 `]`로 둘러싸인 구간은 그룹
- 메타 문자 중 (서로 짝이 맞는) `(`와 `)`로 둘러싸인 구간은 그룹
  (문자열 택일의 좌우 경계를 위해 사용한 `(`와 `)`도 포함)

반복 횟수 표현

정규식에서 앞 그룹이 연이어 반복되는 횟수를 나타내는 특수 기호들 (단, n과 m은 자연수)

기호	의미	예시	
		정규식[7]	대응하는 문자열들
`?`	0 번 혹은 1 번	`ab?c`	ac, abc
`*`	0 번 이상	`ab*c`	ac, abc, abbc, abbbc, abbbbc, ...
`+`	1 번 이상	`ab+c`	abc, abbc, abbbc, abbbbc, ...
`{n}`	n 번	`ab{3}c`	abbbc
`{n,m}`	n 번 이상 m 번 이하[8]	`ab{1,3}c`	abc, abbc, abbbc
`{n,}`	n 번 이상	`ab{2,}c`	abbc, abbbc, abbbbc, ...

사실상 동치인 것들:

- `ab?c` = `ab{0,1}c` = `a(¦b)c` = `(ac¦abc)`
- `ab*c` = `ab{0,}c`
- `ab+c` = `ab{1,}c` = `abb*c`

[7] 문서에 기재할 때에는 이렇게 윗첨자로 표시하는 경우가 많은데, 어디까지나 가독성을 위한 관례일 뿐 필수 사항은 아님.

[8] "m번 이하"를 의미할 만한 `{,m}`를 따로 만들지 않은 이유는 그것이 `{0,m}`에 해당하기 때문임

택일을 “반복”한다는 의미

택일한 ‘결과’를 반복하는 게 아니라 ‘택일 자체’를 반복하는 것!

[예시] 통장 비밀번호는 십진법 숫자 4 자리: 정규식 “`[0-9]{4}`”로 표현됨

→ 위 정규식은 “`[0-9][0-9][0-9][0-9]`” 와 완전히 같은 의미

→ 각 `[0-9]`마다 어떤 숫자를 택일하든, 그 결과가 다른 세 택일에 일체 영향을 주지 않음 (서로 별개).

→ 만일 `[0-9]{4}`가 택일한 결과를 네 번 반복한다는 의미였다면, 각 자리 숫자는 모두 서로 같다는 뜻이 됨 (0000, 1111, 2222, ..., 8888, 9999 만 대응)

[예시] 문자열 택일도 마찬가지: “`(me & |you )*`” → “`me & me & me & me & you me & me & ...`”

[예시] 아무거나 (`.`) 택일도 마찬가지: “`.*`”는 (빈 문자열을 포함하여 아무 길이의, 개행문자만 제외된) 그야말로 아무 문자열

[예시] 선행 0 없이 표기된 0 이상 9999 이하 범위의 정수: 정규식 “`(0|[1-9][0-9]{0,3})`”로 표현됨

→ 위 정규식은 “`(0|[1-9]|[1-9][0-9]|[1-9][0-9][0-9]|[1-9][0-9][0-9][0-9])`” 와 완전히 같은 의미

→ 만일 택일한 결과를 반복하는 것이었다면, d 자리 정수의 뒤쪽 d-1 개 숫자들은 모두 서로 같은 숫자

역방향 Escaping: 리터럴 문자 → 메타 문자

Q: 항상 일일이 “`[0123456789]`” 혹은 “`[0-9]`”라고 입력해야 하나요?

A: 사실은 그냥 “`\d`”라고 할 수도 있습니다…만, (적어도 이 수업에서는) **추천하지 않습니다**.

평범한 리터럴 문자에 백슬래시가 붙으면 메타 문자가 되는 경우 있음

- `\d` 숫자 (`[0-9]`와 동일…이 아니고 사실은 전각 숫자 등 훨씬 많은 종류의 숫자들이 포함됨)
- `\D` 숫자가 아닌 문자 (`[^0-9]`와 동일…이 아니고 사실은 전각 숫자 등 훨씬 많은 종류의 숫자들이 제외됨)
- `\s` 공백들 (Perl에서는 `[ \t\r\n\v\f]`, Vim에서는 `[ \t]`)
- `\S` 공백이 아닌 문자들
- `\w` 알파벳, 숫자, 밑줄 (`[A-Za-z0-9_]`에 전각 로마자 등등 다양하게 포함)
- `\W` 알파벳, 숫자, 밑줄이 아닌 문자들 (`[^A-Za-z0-9_]` 외에도 많이 제외됨)

문장의 처음(^)과 끝($)에 대한 부연

대부분의 언어 및 도구에서 기본적으로 **부분 문자열** 대응

- 정규식이 주어진 문자열 중 일부분을 표현하고 있으면 “대응한다”고 답함
- 문자열 전체를 표현하는지 확인하려면 정규식 앞 뒤에 각각 “^”와 “$” 필요
- 텍스트 파일 내용 검색이나 치환 작업 등에 편리 (일부 문자열 규칙만 표현)

반면, 일부 언어나 도구에서는 기본적으로 전체 문자열 대응

- 정규식이 주어진 문자열 전체를 표현하고 있어야만 “대응한다”고 답함
- 문자열 일부에 대응하는지 확인하려면 정규식 앞뒤에 “`.*`” 필요 (예: “`.*cde.*`”)
- 키입력 문자열 형식 검사 등에 편리 (대체로 입력 문자열 전체 규칙 표현)

이 과목에서는 기본적으로 **전체 문자열** 대응을 추천

- 대체로 키입력 형식 검사가 주류일 것으로 예상
- 이 강의 슬라이드도 전체 문자열 대응을 기준으로 작성
- 강제는 아님. 본인들 판단에 따라 정하되, 변경 시 문서에 명시할 것

정규 표현식	대응 문자열들
`cde`	cde abcde cdefg abcdefg
`^cde`	cde cdefg
`cde$`	cde abcde
`^cde$`	cde

Table 1: 부분 문자열 대응

아주 약간의 예시들

[예시] 휴대전화 번호: `01[01](-[0-9]{4})(2)`

→ 문제점: 중간 자리는 010의 경우 4자리 뿐이지만, 011의 경우 3자리도 있고 4자리도 있음

→ 수정 반영 #1: `01(0-[0-9]{4}|1-[0-9]{3,4})-[0-9]{4}`

→ 수정 반영 #2: `01(0-[0-9]|1-[0-9]?)[0-9]{3}-[0-9]{4}`

[예시] 5분 단위로만 입력 가능한 24시간제 시간 형식: `([01][0-9]|2[0-3]):[0-5][05]`

[예시] 학점 등급: `([A-D]\+?|P|F)`

[예시] 방명록 데이터 파일 형식: `([가-힣]{2,5}\t[^\t\n]*\t[1-9][0-9]*\n)*`

- 전체 파일은 여러 개의 (서로 개행 문자로 구분된) 레코드로 구성됨
- 각 레코드는 (서로 탭 문자로 구분된) 이름 필드, 메모 필드, 연번 필드로 구성됨
- 이름은 한글로 2 글자 이상 5 글자 이하
- 메모는 필드 구분자인 탭문자와 레코드 구분자인 개행문자를 제외한 아무 문자로 0개 이상 임의의 길이
- 연번 필드는 선행 0을 허용하지 않는 양의 정수

정규 표현식으로 못 하는 것

Q1: 문자 a가 n번 있고 그 뒤에 문자 b가 똑같이 n번 있는 문자열들은 정규식으로 어떻게 표기하나요? 그러니까, “” (빈 문자열), “ab”, “aabb”, “aaabbb”, “aaaabbbb” 이런 것들 모두에(만) 대응하는 정규식을 어떻게 표기하나요?

Q2: 산술 수식처럼 재귀적인 문법은 정규식으로 어떻게 표기하나요?

A: **못 해요. ^_^**

- 정규식은 원래 유한 상태 기계가 인지하는 언어인 정규 언어를 표현하기 위한 표기법
- 유한 상태 기계로 풀 수 없는 문제에 해당하는 언어는 정규식으로 표현 못 함
- 유한 상태 기계에는 ‘기억 장치’가 없음 (정규식 확장 기능에 기억 기능이 들어가면 정규식이 아닌 이유)

Q: 그 답이 최선인가요?

A: 다음 수업에서 다룰 BNF 문법으로 표현할 수 있습니다. 다음 수업을 기대하세요.