テキスト処理 第9回 (2008-06-17)

田中哲
産業技術総合研究所
情報技術研究部門
akr@isc.senshu-u.ac.jp
http://staff.aist.go.jp/tanaka-akira/textprocess-2008/

今日の内容

- レポートの解説
- cap_include と cap_exact
- match_include と match_exact
- 正規表現エンジンにいくつか機能を拡張する (再帰を使わない機能)
- レポート

cap_include

- 文字列内で正規表現がマッチする部分文字列を探す
- 見つかったらそのときのキャプチャ情報を返す
- キャプチャ情報には :all としてマッチ全体の位置を入れる
- 見つからなければ nil を返す

```
def cap_include(r, str)
  str = str.split(//)
  0.upto(str.length) {|b|
    try(r, str, b, {}) {|e, md|
      md = md.dup
      md[:all] = b...e
      return md
    }
  }
  nil
end
```

cap_include の実行例

- p cap_include([:cat, "a", "p"], "pineapple")
  #=> {:all=>4...6}

- p cap_include([:cat, [:rep, "a"], "p"],
  "pineapple")
  #=> {:all=>0...1}

- p cap_include(
  [:cat, [:capture, :a, [:rep, "a"]], "p"],
  "pineapple")
  #=> {:a=>0...0, :all=>0...1}

cap_exact

- 文字列全体が正規表現にマッチするか試す
- マッチするならそのときのキャプチャ情報を返す
- キャプチャ情報には :all としてマッチ全体の位置を入れる
(あまり有用ではない)
- 見つからなければ nil を返す

```
def cap_exact(r, str)
  str = str.split(//)
  try(r, str, 0, {}) {|e, md|
    if str.length == e
      md = md.dup
      md[:all] = 0...e
      return md
    end
  }
  nil
end
```

match_include

- 文字列内で正規表現がマッチする部分文字列を探す
- 見つかったらその部分文字列の位置を返す
- 見つからなければ nil を返す

```
def match_include(r, str)
  str = str.split(//)
  0.upto(str.length) {|b|
    try(r, str, b, {}) {|e, md|
      return b...e
    }
  }
  nil
end
```

match_include の実行例

- p match_include([:cat, "n", "g"], "orange")
  #=> 3...5
- p match_include([:cat, "e", [:rep, "r"]],
  "berry")
  #=> 1...4
- p match_include([:anychar], "z")
  #=> 0...1

match_exact

- 文字列全体が正規表現がマッチするか試す
- マッチするならマッチした位置を返す
- 見つからなければ nil を返す

```
def match_exact(r, str)
  str = str.split(//)
  try(r, str, 0, {}) {|e, md|
    if str.length == e
      return 0...e
    end
  }
  nil
end
```

いくつか機能を拡張する

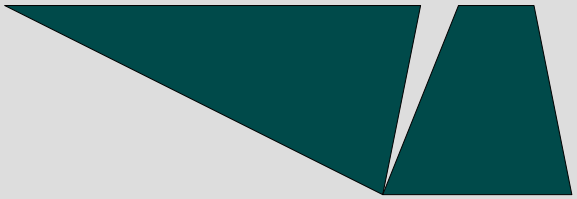
- ¥A　文字列の先頭の位置
- ¥z　文字列の末尾の位置
- ^　行頭
- $　行末

文字列の先頭の位置 ¥A

- 文字列の最初にマッチするパターン [:string_start]
- Ruby の正規表現では ¥A
- このパターン自身は文字を消費しない
  先頭の文字の直前にマッチする
- match_include([:string_start], "abc") #=> 0...0
- match_include([:string_start], "")　　#=> 0...0
- match_include([:cat, [:string_start], "a"], "abc")
  #=> 0...1
- match_include([:cat, [:string_start], "b"], "abc")
  #=> nil

「文字列の先頭の位置」のマッチ

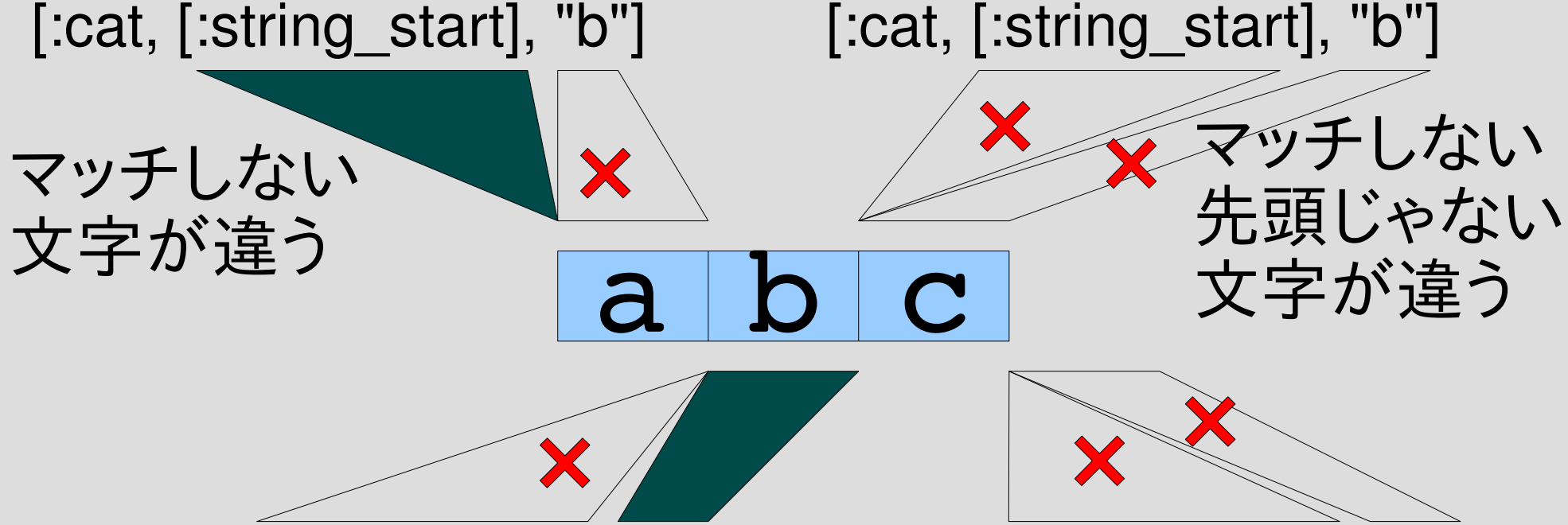
[:cat, [:string_start], "a"]　　マッチする



a	b	c	d	e	f

「文字列の先頭の位置」のマッチ

match_include([:cat, [:string_start], "b"], "abc")



「文字列の先頭の位置」の実装

```
def try(exp, seq, pos, md, &b)
  ...
  when :string_start
    yield pos, md if pos == 0
  ...
end
```

「文字列の先頭」は「pos == 0」

empstr との比較

```
when :empstr
  yield pos, md
when :string_start
  yield pos, md if pos == 0
```

位置の条件がない

位置の条件がある

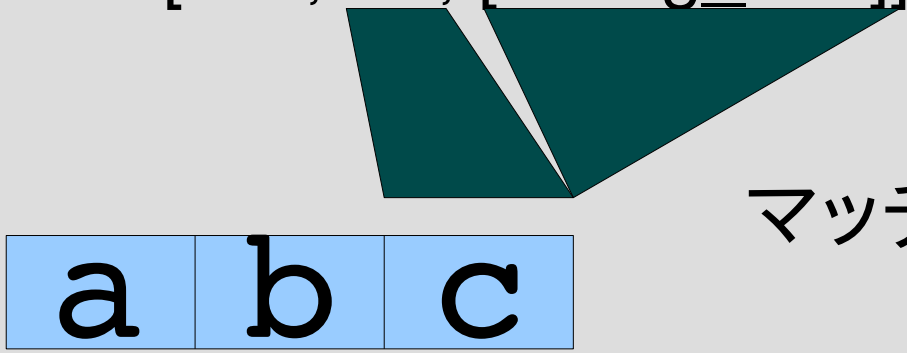
文字列の末尾の位置 ¥z

- 文字列の最後にマッチするパターン [:string_end]
- Ruby の正規表現では ¥z
- このパターン自身は文字を消費しない
  末尾の文字の直後にマッチする
- match_include([:string_end], "abc") #=> 3...3
- match_include([:string_end], "") #=> 0...0
- match_include([:cat, "c", [:string_end]], "abc")
  #=> 2...3
- match_include([:cat, "b", [:string_end]], "abc")
  #=> nil

「文字列の末尾の位置」のマッチ

match_include([:cat, "c", [:string_end]], "abc")

[:cat, "c", [:string_end]]



マッチする

「文字列の末尾の位置」の実装

```
def try(exp, seq, pos, md, &b)
  ...
  when :string_end
    yield pos, md if pos == str.length
  ...
end
```

「文字列の末尾」は
「pos == str.length」

empstr との比較

```
when :empstr
  yield pos, md
```

位置の条件がない

```
when :string_end
  yield pos, md if pos == str.length
```

位置の条件がある

match_include で match_exact を実現

- string_start と string_end を使えば match_exact は match_include で実現できる

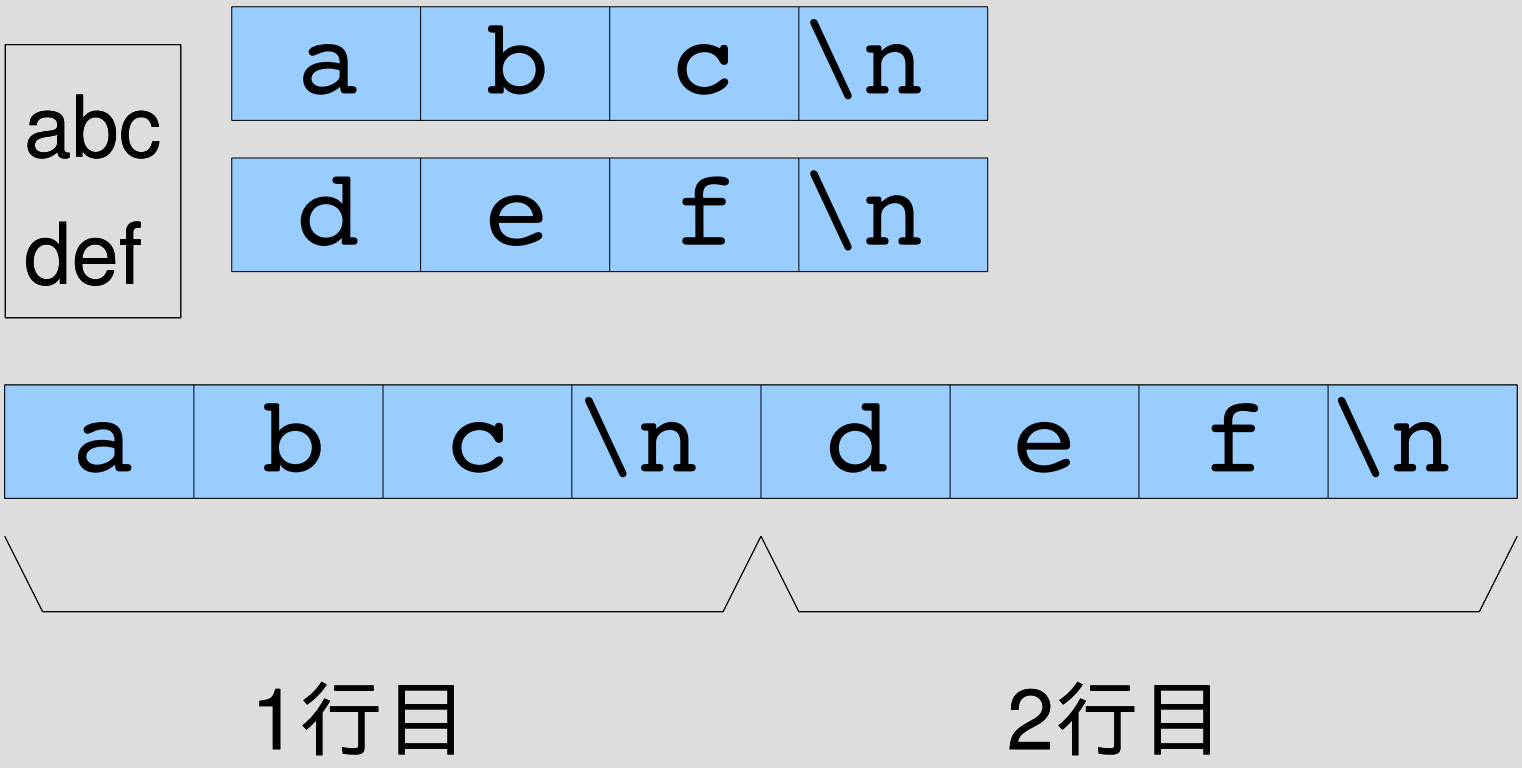
```
def match_exact(r, str)
  match_include(
    [:cat, [:string_start],
           r,
           [:string_end]],
    str)
end
```

行頭 ^

- 行の先頭にマッチするパターン [:line_start]
- Ruby の正規表現では ^
- このパターン自身は文字を消費しない
- 文字列の先頭と改行の直後にマッチする

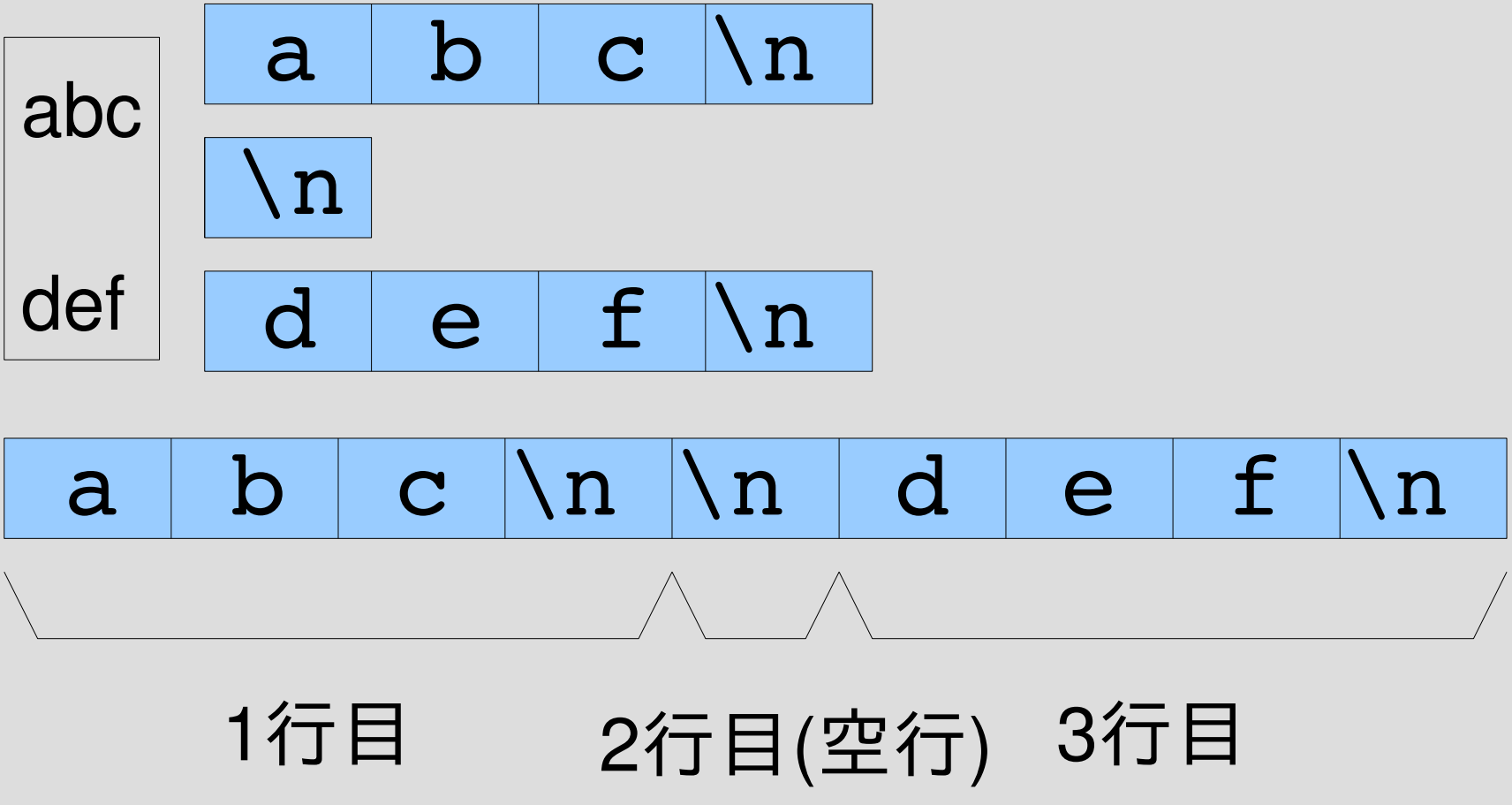
行の構造

- 文字列は文字の並び
- 文字のひとつに改行文字 ¥n というものがある
- 改行文字で行の終端を表す



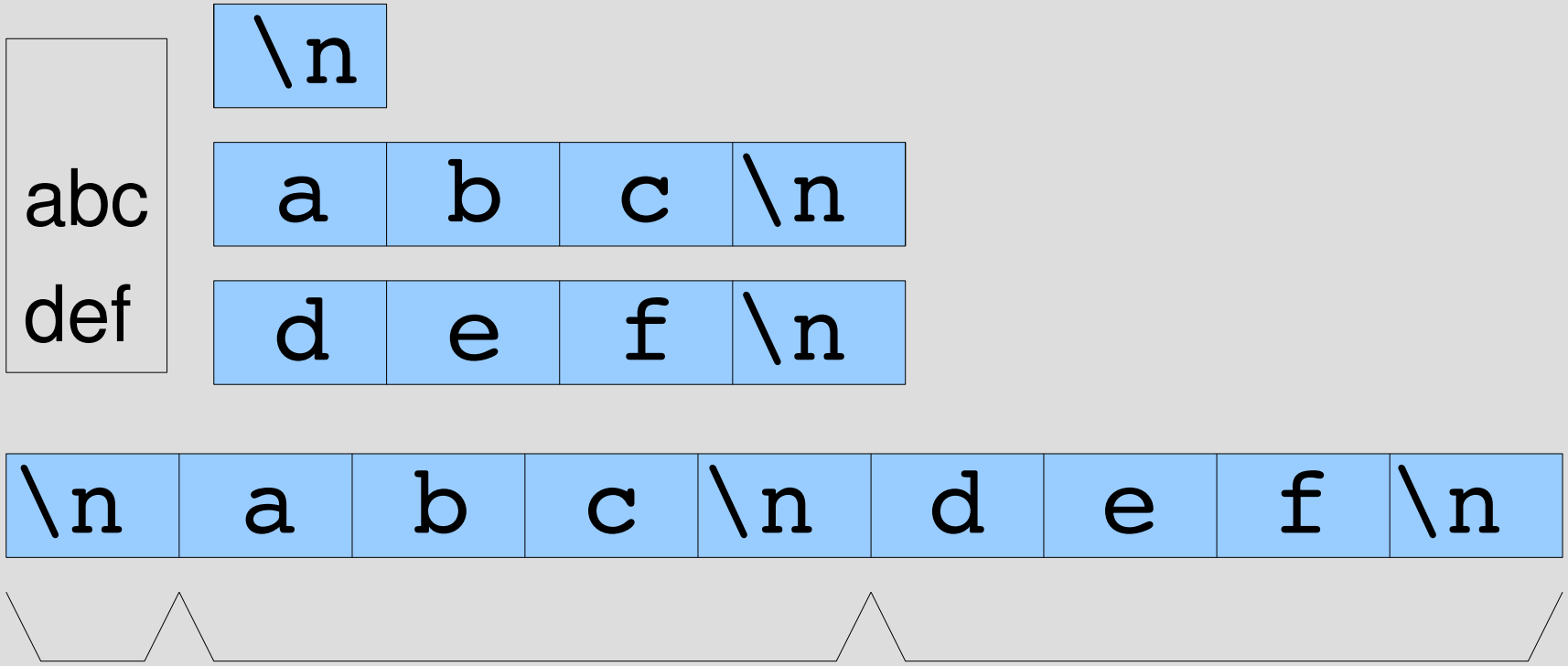
空行

- 改行以外に文字がない行が空行
- ファイル途中に空行があると改行が連続する



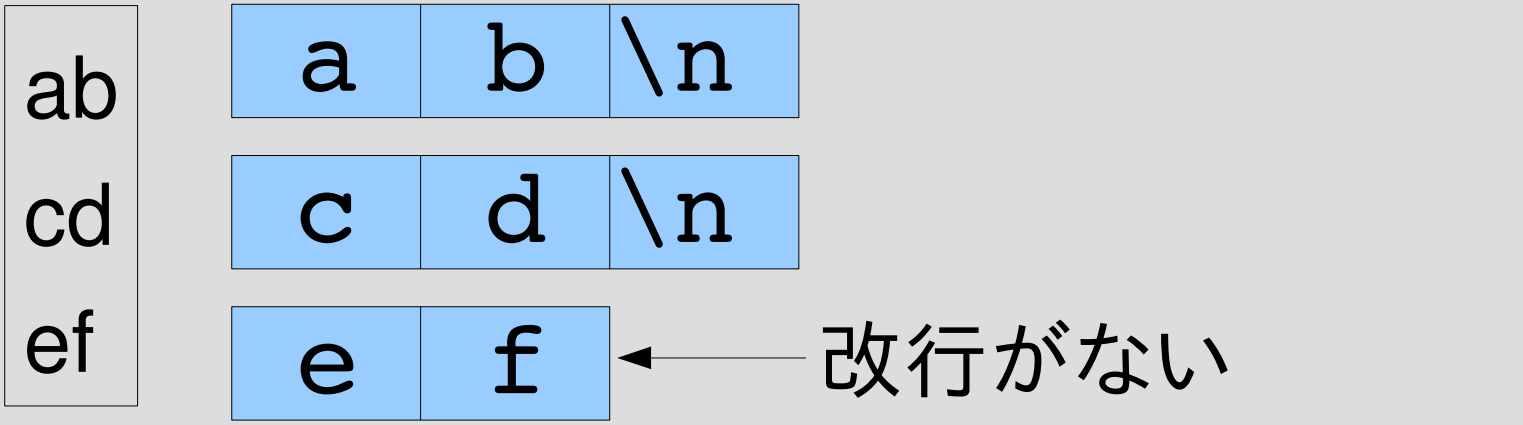
空行 (文字列先頭)

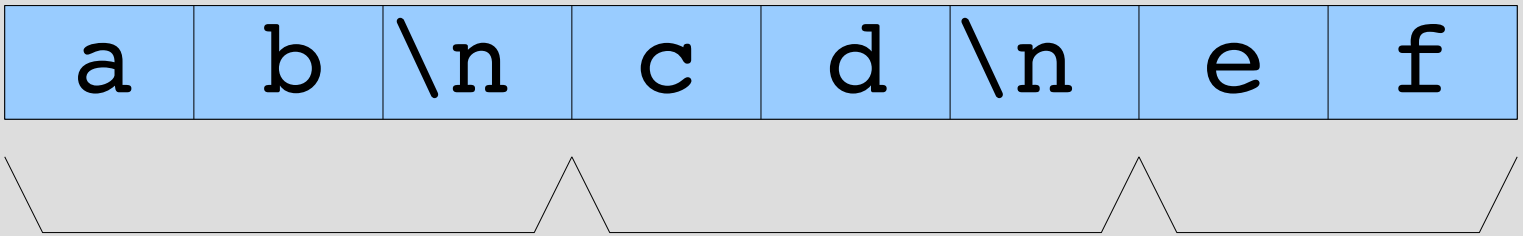
- 改行以外に文字がない行が空行
- 文字列の先頭に改行



不完全な行

- 文字列末尾の改行で終わらない行
- 文字列が改行で終わっているときには行がない とみなす (この場合不完全な行は存在しない)





1行目　　2行目　　3行目(不完全な行)

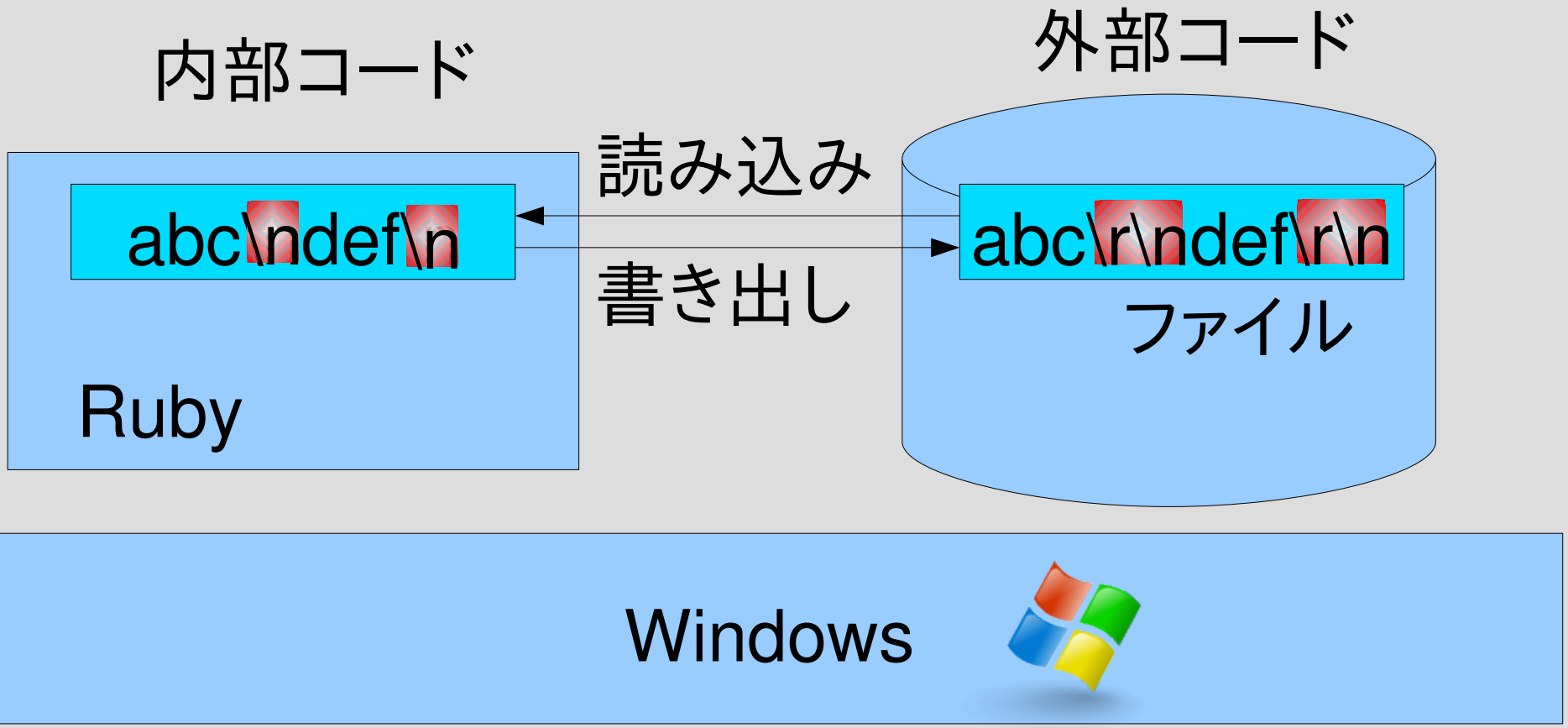
空文字列

- 空文字列 "" に行は入っているか？
- おそらく入っていないと考えるのが自然
  - 入っているとしたら、何行入っている？
- でも、伝統的に、行頭、行末は空文字列にマッチする
- この講義では伝統に従う

改行は環境依存

- 改行の標準的な表現は OS によって違う
  - Windows では¥r¥nの 2文字の並びがひとつの改行
  - Unix では¥nが改行
  - 昔のMacintoshでは¥rだった
  - Unicode にはIBMの大型機由来のとかも入っている
- 違いをいちいち気にするのは面倒なので、
  読み込むときに Unix の形式に変換し、
  書き出すときに環境依存の改行に変換する

内部コードと外部コード



テキストモードとバイナリモード

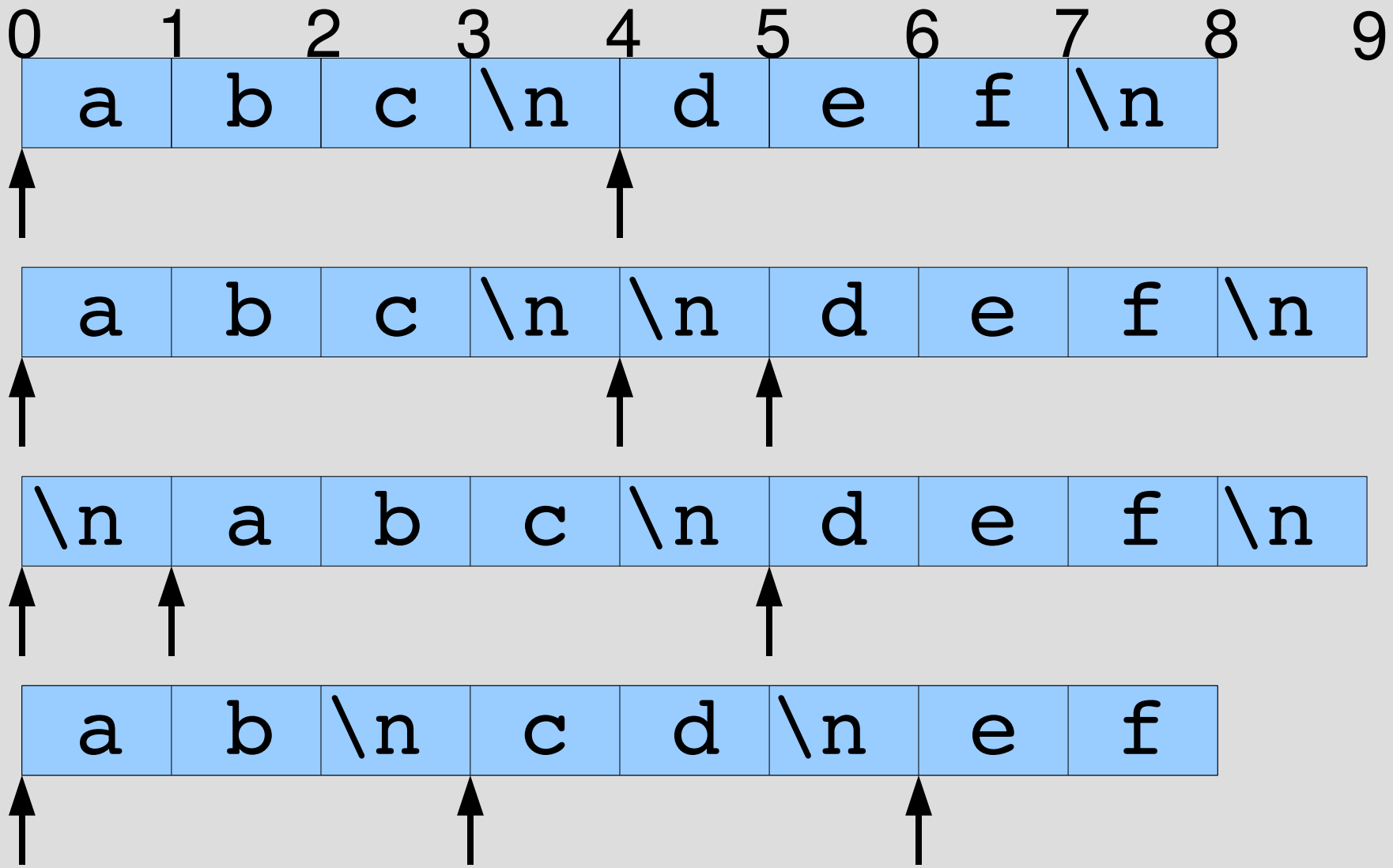
- 画像など、行構造ではないファイルを扱うときに変換が行われるとデータが壊れて困る
- 変換するかどうかのモードがある
- デフォルトは変換するテキストモード

- テキストモード
  - ファイルの読み書きで改行の変換を行う
- バイナリモード
  - ファイルの読み書きで改行の変換を行わない

行頭 ^

- 行の先頭にマッチするパターン [:line_start]
- Ruby の正規表現では ^
- このパターン自身は文字を消費しない
- 文字列の先頭と文字列末尾でない改行の直後にマッチする
- 改行は変換済みで、¥n になっているとする
- match_include([:line_start], "abc")
  #=> 0...0
- rx_ends(
  [:cat, [:rep, [:anychar]], [:line_start]],
  "a¥nb¥n", 0) #=> [2,0]

行頭



「行頭」の実装

```
def try(exp, seq, pos, md, &b)
  ...
  when :line_start
    if pos == 0 ||
       (pos < str.length && str[pos-1] == "\n")
      yield pos, md
    end
  ...
end
```

string_start との比較

```
when :string_start
  yield pos, md if pos == 0

when :line_start
  if pos == 0 || (pos < str.length && str[pos-1] == "\n")
    yield pos, md
  end
```

条件が緩くなっている

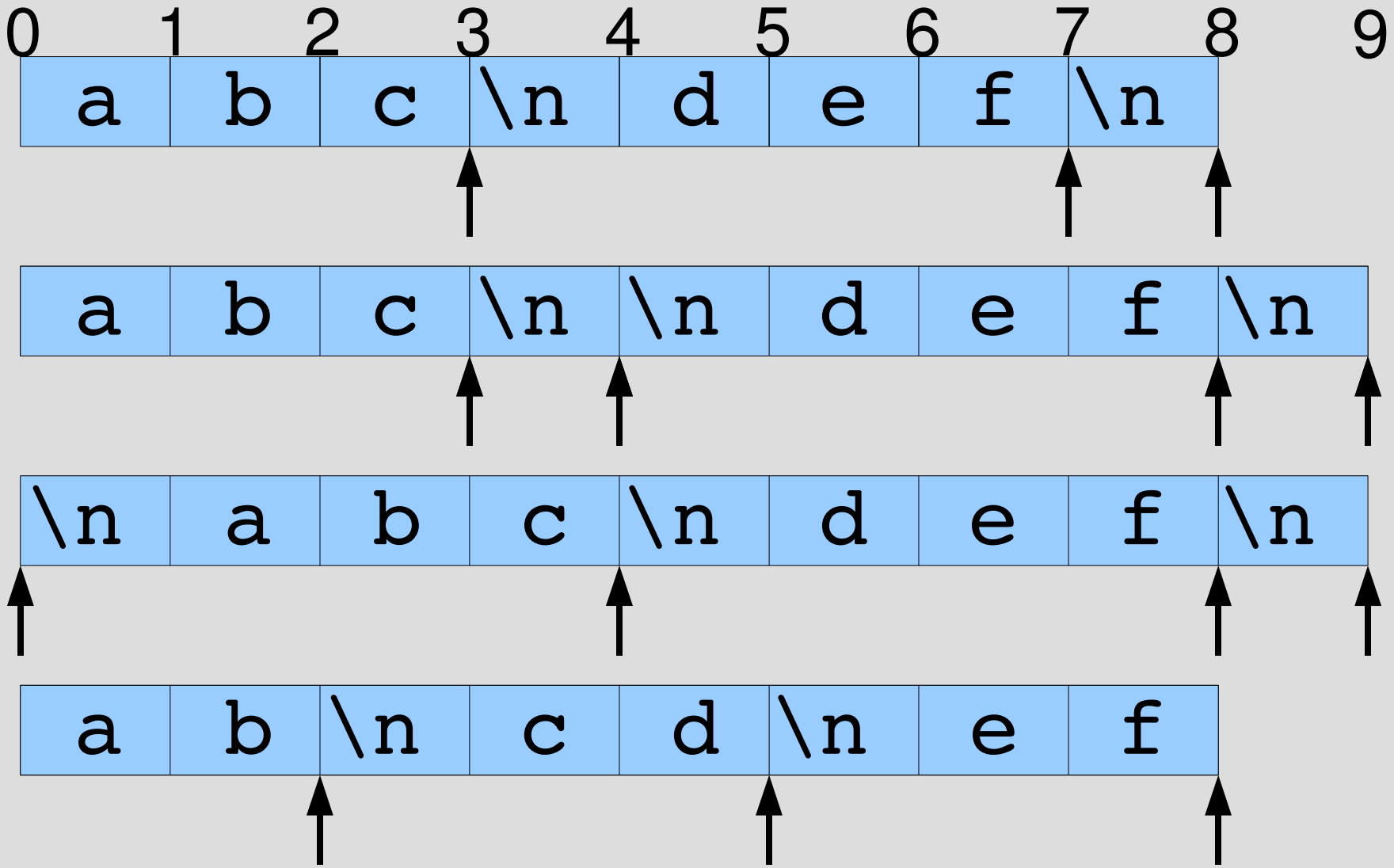
文字列の末尾でない

直前の文字が\n

行末 $

- 行の末尾にマッチするパターン [:line_end]
- Ruby の正規表現では $
- このパターン自身は文字を消費しない
- 文字列の最後と改行の直前にマッチする
- 文字列が改行で終わっていても文字列の最後にマッチする (歴史的慣習)

行末



「行末」の実装

```
def try(exp, seq, pos, md, &b)
  ...
  when :line_end
    if pos == str.length || str[pos] == "\n"
      yield pos, md
    end
  ...
end
```

string_end との比較

```
when :string_end
  yield pos, md if pos == str.length

when :line_end
  if pos == str.length || str[pos] == "\n"
    yield pos, md
  end
```

文字列の末尾

条件が緩くなっている

直後の文字が\n

レポート

- cap_exact を用いて match_include を定義せよ
- そうやって実装した match_include と講義で説明したものに挙動の違いが有れば具体例をあげて違いを述べよ

- ユニットテストを提供するので、実装したらテストして確認すること
- ∠切 2008-06-24 12:00
- RENANDI
- 拡張子が txt なテキストファイルを望む

match_include の実行例

- p match_include([:cat, "n", "g"], "orange")
  #=> 3...5
- p match_include([:cat, "e", [:rep, "r"]],
  "berry")
  #=> 1...4
- p match_include([:anychar], "z")
  #=> 0...1

まとめ

- 前回のレポートの解説
- cap_include, cap_exact, match_include, match_exact を定義
- いくつか機能拡張
  - [:string_start]
  - [:string_end]
  - [:line_start]
  - [:line_end]
- レポートを出した