I117（11）CSV パース

知念

北陸先端科学技術大学院大学 情報科学研究科
School of Information Science,
Japan Advanced Institute of Science and Technology

CSV RFC4180

- CSV (Comma-Separated Values)
  - コンマで区切ったテキストファイル
  - スプレッドシートプログラムで良く使われる
- RFC4180
  - 長年明解な規約がなかったが、2005年にRFCとなる
  - http://www.ietf.org/rfc/rfc4180.txt

例

header つき

```
field1,field2,field3 CRLF
aaa,bbb,cccc CRLF
zzz,yyy,xxxx CRLF
```

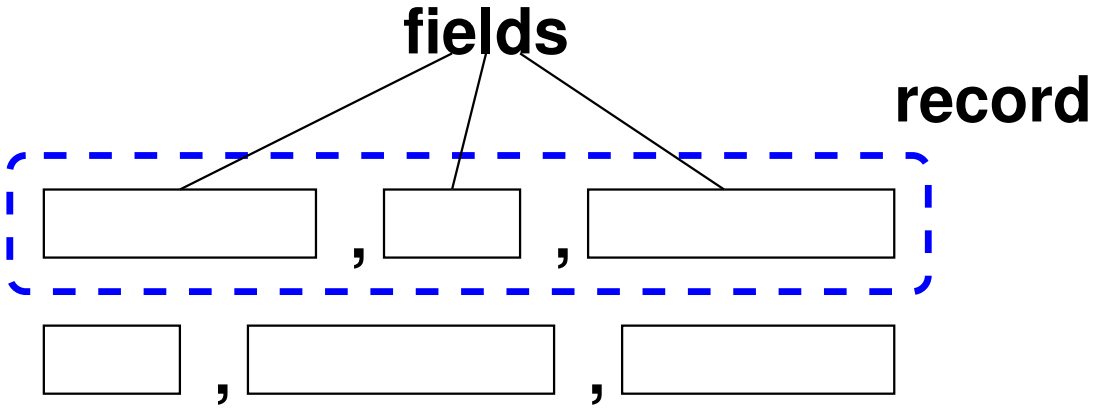
header なし

```
aaa,bbb,cccc CRLF
zzz,yyy,xxxx CRLF
```

ひとまず header なしですすめる

用語

- （原則）行をレコード、コンマで区切られた文字列をフィールドと呼ぶ



- コンマをデータとして扱うには？
  - ダブルクォート(0x22)で文字列を囲む
  - 文字列にはコンマや改行を含めて良い

用語 *(cont.)*

- ダブルクォートをデータとして扱うには？
  - もう一つダブルクォートをつける

難しい例
以下は2フィールドのレコードが3つ

```
"abc,def",ghi CRLF
abc def,"ghi CRLF
xyz" CRLF
"foo""bar",junk dummy CRLF
```

RFC 上の定義

```
file = [header CRLF] record *(CRLF record) [CRLF]
header  = name *(COMMA name)
record  = field *(COMMA field)
name    = field
field   = (escaped / non-escaped)
escaped = DQUOTE
        *(TEXTDATA / COMMA / CR / LF / 2DQUOTE)
        DQUOTE
non-escaped = *TEXTDATA
COMMA = %x2C    DQUOTE = %x22
CR = %x0D       LF = %x0A       CRLF = CR LF
TEXTDATA = %x20-21 / %x23-2B / %x2D-7E
```

ファイル内容確認 pcrlf.c

```
while(fgets(line, BUFSIZ, stdin)) {
  p = line;
  while(*p) {
    if(*p=='\r') {
      if(*(p+1)=='\n') { printf(" CRLF"); p++;  }
      else {             printf(" CR"); } }
    else if(*p=='\n') {  printf(" LF"); }
    else {               fputc(*p, stdout); }
    p++;
  }
  fputc('\n', stdout);
}
```

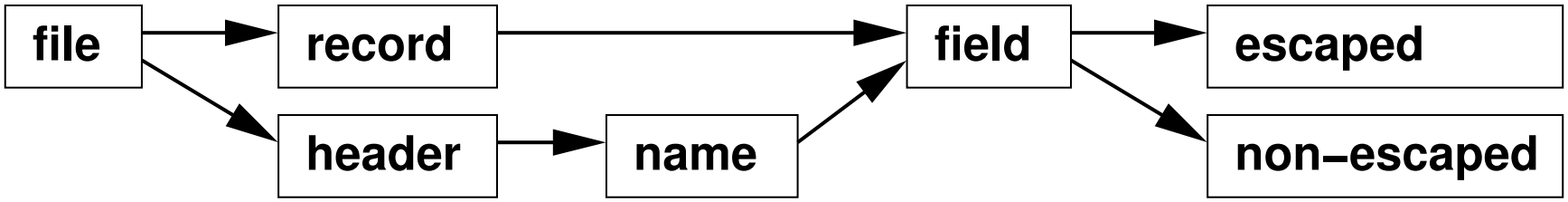
パース csvparse.c

CSV を解釈するプログラムを作る
講義のWWWページに記載
http://www2.jaist.ac.jp/is/private/lecture/i117/2008/csvparse.c

現状では、レコードとフィールド位置、そしてカレントのフィールド内容を格納するだけ

パース csvparse.c *(cont.)*

定義から入力を想像しながら構造を考える



- escaped, non-escape の切替えには入力が必要
- readfield() で新しい入力をはじめる
  - readch() 呼び出し

パース csvparse.c *(cont.)*

escaped 中の 2DQUOTE が難しい

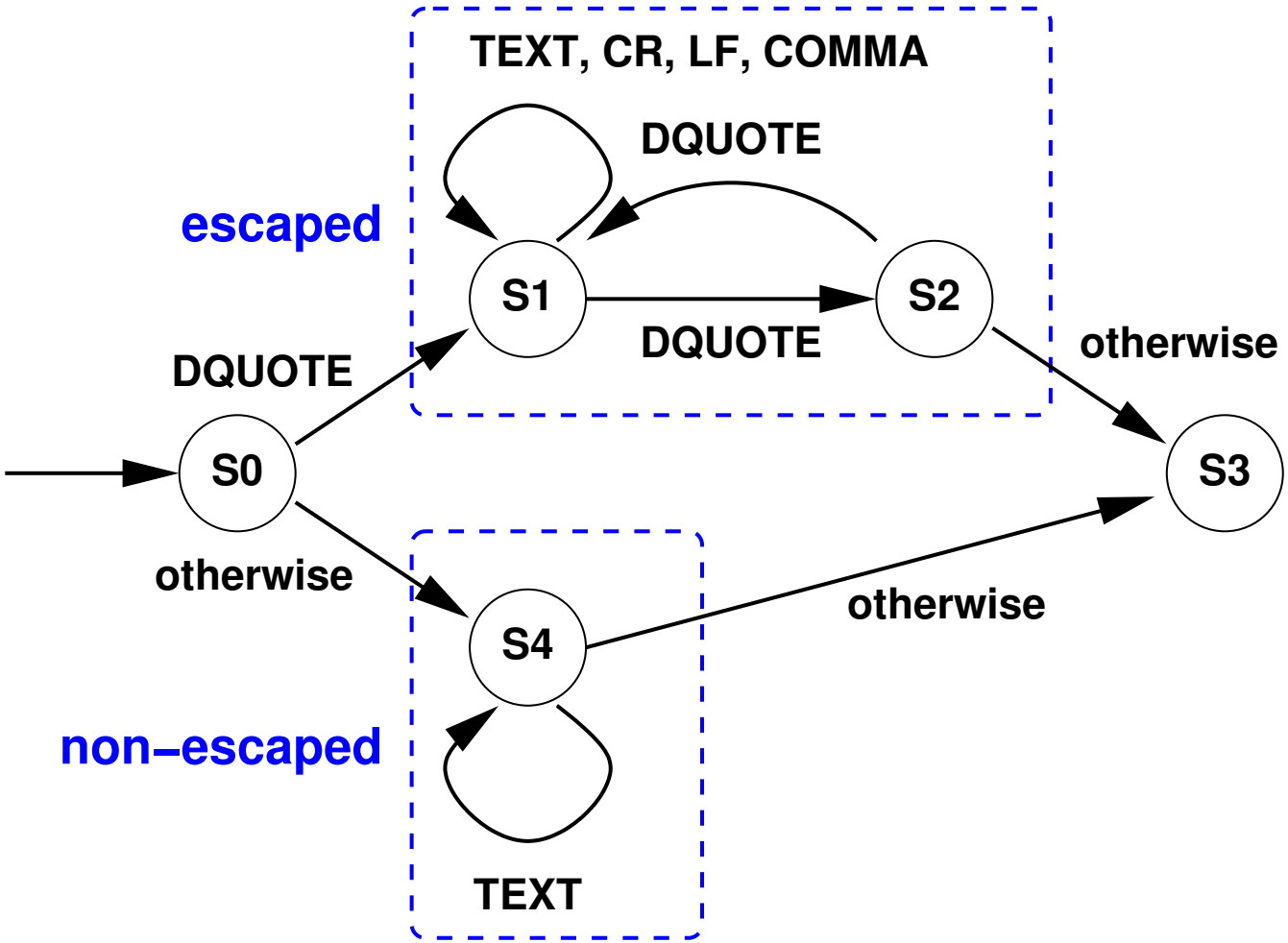
- 2文字読まないと判定できない

それ以外はその場で判定可能

```
#define ISCOMMA  (ch==0x2c)
#define ISCR     (ch==0x0d)
#define ISDQUOTE (ch==0x22)
#define ISLF     (ch==0x0A)
#define ISTEXT   ((ch>=0x20&&ch<=0x21)|| \
   (ch>=0x23&&ch<=0x2b)||(ch>=0x2d&&ch<=0x7e))
```

多文字入力は大変なので1文字入力で設計する

field の状態遷移


TEXT, CR, LF, COMMA
DQUOTE
escaped
S1
S2
DQUOTE
otherwise
DQUOTE
S0
S3
otherwise
otherwise
S4
non–escaped
TEXT

2DQUOTE まわり

- escaped の中では DQUOTE の後は DQUOTE か COMMA に限られる
- DQUOTE が二つでてこなかったら escaped が終ったとみなして良い
- 関数の最後に goto 文でジャンプ

```
int readescaped() {
  readch();
  while(ISTEXT||ISCOMMA||ISCR||ISLF||ISDQUOTE) {
      if(ISDQUOTE) {
          readch();
          if(ISDQUOTE) { /* 2DQUOTE */ }
          else {  goto shiftS3; }
      }
      addch();  readch();
  }
  if(ISDQUOTE) { readch();  }
shiftS3:
  return 0;  }
```

2DQUOTE まわり *(cont.)*

このルーチンは goto を使わないと結構厄介

- 単純に DQUOTE の判定文を抜けると
- escaped を閉じる DQUOTE は読み込んだ後
- つまり読み過ぎていて、閉じる DQUOTE の判定ができない

goto を取り除きたい者は考えてみると良いだろう

ループ — readrecord

素直に作った場合

```
fno = 0;
do {
    ck = readfield();
    if(ISCOMMA) { }
    else
        break;
    fno++;
} while(1);
```

ループ — **readrecord** ***(cont.)***

do-while の形のまま簡単化

```
fno = 0;
do{
    ck = readfield();
    fno++;
} while(COMMA);
```

最後フィールドの後の fno の処理が違うが、今回の範囲では影響はない

ループ — **readrecord** *(cont.)*

fno の処理をそのまま維持して簡単化

```
fno = 0;
while(ck = readfield(), COMMA) {
    fno++;
}
```

readheader や readfile も同様に最適化可能

実行例 — 前述の入力

```
% ./a.out < psmp1b
0-0 field ’abc,def’
0-1 field ’ghi’
1-0 field ’abc def’
1-1 field ’ghi
xyz’
2-0 field ’foo"bar’
2-1 field ’junk dummy’
EOF
```

想定通り

実行例 — 前述の入力 *(cont.)*

入力を確認 — 各行 CRLF が入っている

```
% ./pcrlf <psmp1b
"abc,def",ghi CRLF
abc def,"ghi CRLF
xyz" CRLF
"foo""bar",junk dummy CRLF
```

- UNIX の行区切りは LF
- エディタでコントロールM を入力する
  - emacs: コントロールQ コントロールM
  - vi: コントロールV コントロールM

実行例 — 前述の入力 *(cont.)*

出力を再確認 — CR が含まれる行がある

```
% ./a.out < psmp1b |& ./pcrlf
0-0 field ’abc,def’ LF
0-1 field ’ghi’ LF
1-0 field ’abc def’ LF
1-1 field ’ghi CRLF
xyz’ LF
2-0 field ’foo"bar’ LF
2-1 field ’junk dummy’ LF
EOF LF
```

実行例 — RFC のサンプル

1番目

```
% ./a.out < smp1
0-0 field 'aaa'
0-1 field 'bbb'
0-2 field 'ccc'
1-0 field 'zzz'
1-1 field 'yyy'
1-2 field 'xxx'
EOF
```

2番目

```
% ./a.out < smp2
0-0 field 'aaa'
0-1 field 'bbb'
0-2 field 'ccc'
1-0 field 'zzz'
1-1 field 'yyy'
1-2 field 'xxx'
no LF (record separator
```

実行例 — RFC のサンプル *(cont.)*

6番目

```
% ./a.out < smp6
0-0 field ’aaa’
0-1 field ’b
bb’
0-2 field ’ccc’
1-0 field ’zzz’
1-1 field ’yyy’
1-2 field ’xxx’
no LF (record separator)
```

7番目

```
% ./a.out < smp7
0-0 field ’aaa’
0-1 field ’b"bb’
0-2 field ’ccc’
no LF (record separator
```

まとめ

- 少々複雑な形式を解析する例を紹介した
- 定義からプログラムを想像できるように訓練せよ
  - 状態遷移や入力に注意を払う

補足

- RFC に登場する他の例も動作確認済
- 定義上でとりうる全ての組み合わせを試したわけではない
- 行末は CR がない場合も想定して、LF だけでも受け付ける

演習

前述のプログラムに関して

1) 定義上存在するにもかかわらず受け付けない入力を探せ★
   - 長さに関する制限は除外する
2) 変数 vused を使って変数 value への代入を監視せよ★
   - 現行は非常に長い文字列を代入して記録を破壊する可能性がある
3) goto 文を使わず escaped をパースするよう変更せよ★★

演習 *(cont.)*

新規プログラム

1) 行毎に読み込み解析するプログラムを作れ★★
2) 多文字入力の状態遷移で解析するプログラムを作れ★★
3) lex,yacc 等を使って同等の解析プログラムを作れ★★
4) CSV を HTML に変換するプログラムを作れ★
   - フィールド内容は短い文字列とする
   - コンマやダブルクォートも含めた文字列も扱う