I117（16）文字列（6）split+数式評価

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

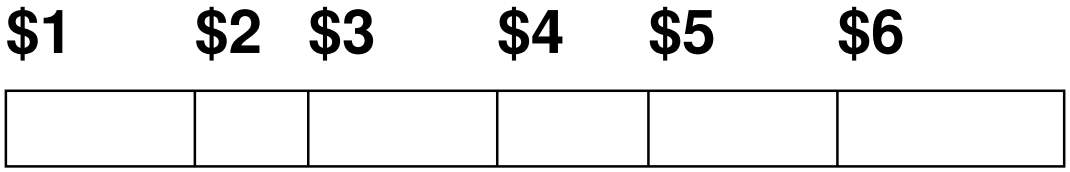
フィールドを含めた数式処理

ファイルの利用はフィールド単位の操作が多い

- あるフィールドの演算
- いくつかのフィールド同士の演算

フィールドを使った数式評価ができると便利

- $ でフィールドを表現
- たとえば、$1+$3 など



フィールドを含めた数式処理 *(cont.)*

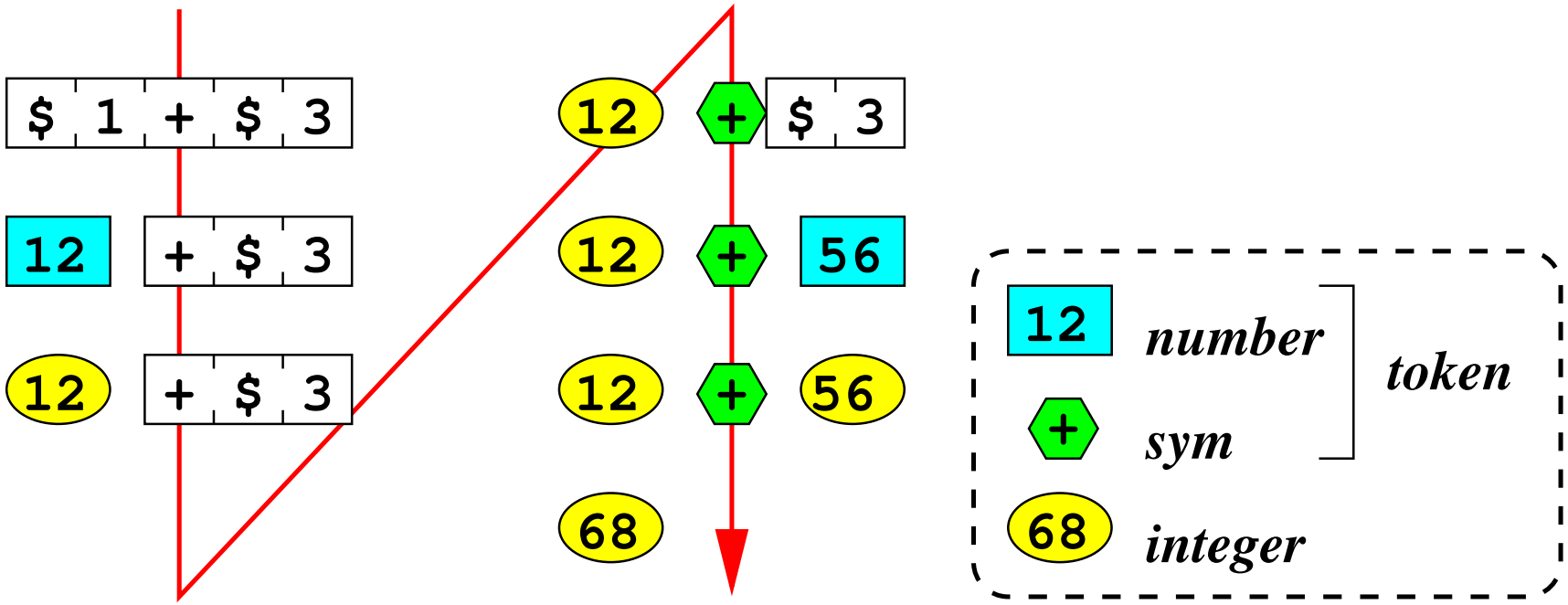
split と数式評価を組み合わせる

- 数式評価プログラムに libspji を読み込む
- レコードを frec とすると $1→frec.field[0]

```
#include "libspji.c"
rec_t frec;
```

- gettokenでフィールドを扱う
  - $ に続く数字がフィールド番号
  - フィールド内容をトークンとする

```
int ap;
...
else if(*pos==’$’) {  pos++;  c = 0;
  while(c<BUFSIZ && *pos>=’0’ && *pos<=’9’) {
      *q++ = *pos++;  c++;  }
  *q = ’\0’;
  tokentype = TNUM;
  ap = atoi(token);
  if(ap<=0 || ap-1>=frec.used) {
    fprintf(stderr,
        "*** ignore field number %d\n", ap);
    return -1;  }
  else { strcpy(token, frec.fields[ap-1]); }  }
```



- 容易に実装可能
- gettoken (字句解析)でフィールド指定を処理するとL1,L2,resolv (構文解析)は変更せず実現できる

メインループ

数式を引数で、ファイルを標準入力で受け取る

```
int main(int argc, char *argv[]) {
  int ck, nf;  char line[BUFSIZ], expr[BUFSIZ];
  strcpy(expr, argv[1]);
  rec_init(&frec);
  while(fgets(line, BUFSIZ, stdin)) {
    chomp(line);
    rec_clear(&frec);
    nf = rec_split(&frec, line, ',');
    ck = Eeval(expr);
  } }
```

実行

データ

```
% cat d1
12,34,56
78,90,1011
1213,1415,1617
```

実行結果

```
% cat d1|./a.out '$1+$3'
68
1089
2830
% cat d1|./a.out '$1*$3'
672
78858
1961421
```

実行 *(cont.)*

即値の数字と混在も可能

```
% cat d1 | ./a.out '$1*2+$2*3'
126
426
6671
```

$演算子

将来、式中で $ を使いたくなるだろう
たとえば...

```
$(1+1)+$3
```

```
x=2
$x+$3
```

講義で登場したプログラムの能力は乏しいが、
前回の演習の成果で変数やカッコは使えるようになっているはず

$演算子 *(cont.)*

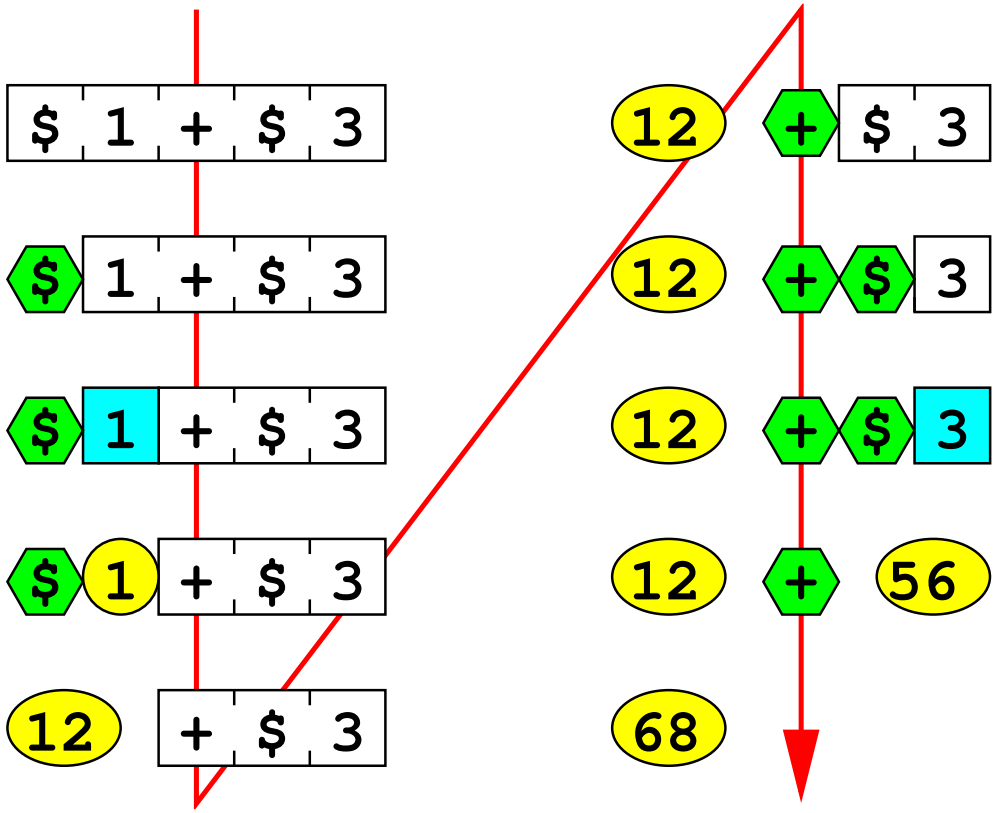
式の中で $ を扱えるようにするため、演算子とする

- OFIELD という演算子を定義する

```
#define OFIELD  '$'
...
        MATCH(OFIELD,   "$",    1);
```

- L3 で OFIELD の演算を処理する
  - 下位の resolv の結果をフィールド番号に
  - その番号でレコードにアクセス

```
int L3(int *rv) {
  int ap, ck;
  if(tokentype==TSYM && tokenop==OFIELD) {
    gettoken();
    ck = resolv(&ap);
    if(ap<=0 || ap-1>=frec.used) {
      fprintf(stderr,
        "*** ignore field number %d\n", ap);
      return -1;
    }
    else { *rv = atoi(frec.fields[ap-1]);  }
    return 0;
  }
  return resolv(rv);  }
```



※ 演算でフィールドを導出できた

gettoken 能力向上

expreval01.c の gettoken は演算子が連続して現れる（単項演算子）ことを想定していない

```
else {
  MATCH(OINC,     "++",    2);
  MATCH(OPLUS,    "+",     1);
  MATCH(OTIMES,   "*",     1);
  if(tokentype==TIG) {
  fprintf(stderr, "***ignore token\n");
  }
}
```

```
else {
   MATCH(OINC,      "++",    2);
   MATCH(OPLUS,     "+",     1);
   MATCH(OTIMES,    "*",     1);
   MATCH(OFIELD,    "$",     1);
   if(tokentype==TIG) {
   fprintf(stderr, "***ignore token\n");
   }
}
```

- 単純に拡張すると
- $1+$2 の入力で +$ のトークンができる
- 直列に処理するため + のあと $ にマッチする

- 見付かった時点で末尾ジャンプすると回避できる

```
else {
  MATCH(OINC,      "++",    2);
  MATCH(OPLUS,     "+",     1);
  MATCH(OTIMES,    "*",     1);
  MATCH(OFIELD,    "$",     1);
  if(tokentype==TIG) {
  fprintf(stderr, "***ignore token\n");
  }
match_done:
  (void)0;
}
```

マクロ MATCH 側には goto 文をいれる

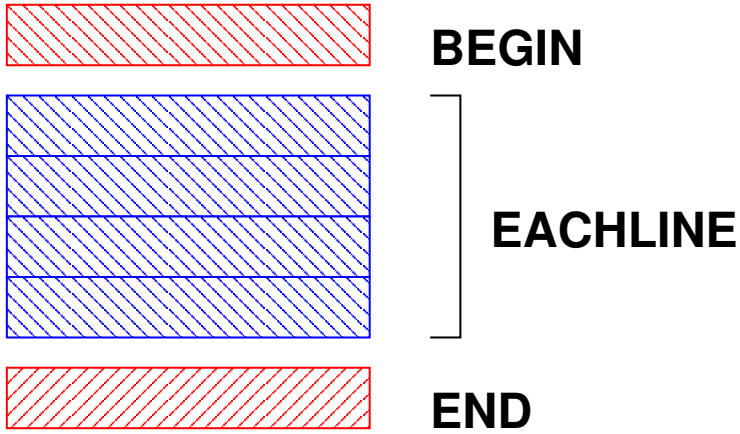
```
#define MATCH(ocode,ostr,olen)  \
  if(*pos==*ostr && strncmp(pos, ostr, olen)==0) { \
      c = olen; \
      while(c>0) { \
            *q++ = *pos++; c--; } \
      *q = '\0'; \
      tokentype = TSYM; \
      tokenop = ocode; \
      goto match_done; }
```

- expreval01b.c として WWWページにおく
- else を使って実現することもできるだろう

データ行以外での処理

データ行以外での数式処理を加える
たとえば、あるフィールドの合計を求める疑似コード

```
s = 0
while(<1行読み込む>) {
     s += $1
}
print s
```



これを以下のように実行する

```
% cat data|./a.out 's=0' 's' 's+=$1'
```

main のみ変更する

```
int main(int argc, char *argv[]) {
  int ck, nf;
  char line[BUFSIZ];

  if(argc<=3) {
    printf("usage: <BEGIN> <END> <EACH-LINE>\n");
    exit(1);
  }
  ck = Eeval(argv[1]);
  if(ck<0) {
    fprintf(stderr, "error in beginning\n");
  }
```

データ行前の式評価

メインループ

```
  rec_init(&frec);
  while(fgets(line, BUFSIZ, stdin)) {
    chomp(line);
    rec_clear(&frec);
    nf = rec_split(&frec, line, ',');
    ck = Eeval(argv[3]);
    if(ck<0) {
        break;
    }
  }
```

これまでと同様

データ行後の式評価

```
  ck = Eeval(argv[2]);
  if(ck<0) {
      fprintf(stderr, "error in ending\n");
  }
  return 0;
}
```

```
% cat d1 | ./a.out 's=0' 's' 's+=$1'
0
12
90
1303
1303
```

データ行以外での処理 *(cont.)*

演算子 += が使えない場合

```
% cat d1 | ./a.out 's=0' 's' 's=s+$1'
```

変数が扱えない場合

```
% cat d1 | ./a.out 111 777 '$1+$3'
111
68
1089
2830
777
```

データ行以外での処理 *(cont.)*

split を使うと複数の式を扱える
データ行前

```
  rec_t bcmd;
  ...
  rec_init(&bcmd);
  rec_split(&bcmd, argv[1], ';');
  for(i=0;i<bcmd.used;i++) {
    ck = Eeval(bcmd.fields[i]);
    if(ck<0) { break;  }
  }
```

```
  rec_t eachcmd;
  ...
  rec_init(&eachcmd);
  rec_split(&eachcmd, argv[3], ';');
  while(fgets(line, BUFSIZ, stdin)) {
    chomp(line);
    rec_clear(&frec);
    nf = rec_split(&frec, line, ',');
    for(i=0;i<eachcmd.used;i++) {
      ck = Eeval(eachcmd.fields[i]);
      if(ck<0) { goto loopend;  }
    } }
loopend:
```

```
rec_t ecmd;
rec_init(&ecmd);
rec_split(&ecmd, argv[2], ';');
...
for(i=0;i<ecmd.used;i++) {
    ck = Eeval(ecmd.fields[i]);
    if(ck<0) {
        break;
    }
}
```

複数式を扱えている

```
% cat d1 | ./a.out 's=0;u=0' 's;u' 's+=$1;u+=$1+$3'
0
0
12
68
90
1157
1303
3987
1303
3987
```

演算子や変数が使えない場合

```
%cat d1|./a.out ’111;222’ ’777;888’ ’$1+$3;$1+$2’
111
222
68
46
1089
168
2830
2628
777
888
```

ここまでくると

- これまでのプログラムを組み合わせるといろいろな処理ができる
- 演習をこなしていれば、これらを自作できる力がついている
- 自力で作れることは非常に重要
  - 製品開発では他プログラムを使えない場面も
    - プラットホームの制限
    - 顧客の要望
    - 知的財産の混入防止
  - 特にフリーソフトやオープンソフトは微妙

演習

1) フィールド内容を数式評価に組み込むプログラムを実際に作成せよ★
2) 文字列を扱える数式評価プログラムに、同様にフィールド内容の処理を組み込め★★

   たとえば、文字列の連結演算子をドットとして、以下のような指示で二つのフィールドを連結して出力するようにせよ

```
% cat datafile |yourprogram '$1.$2'
```