I117（27）数式評価 配列

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

配列

値を複数扱う手段として配列を実現する

```
a[3]=7
a[2]+3
a[3+4]=9
```

これまでのパース方法では単純に扱えない

- これまでは変数名の隣がイコールだった
- ここでは凝った技法を使うことにする

パースの困難さ

これまでのパース方法【LL(1)】では単純に扱えない
たとえば、L0 でイコールが出てくるルールは

```
L0 ::= L1 '=' L1
```

L1 から変数への導出

```
L1 → L2 → L3 → resolv → VAR
```

組み合わせると、変数名の直後イコールが出てくるので、L0 内部で L1 を呼び出した後のイコールで代入を判別できる

```
L0 ::= VAR '=' L1
```

パースの困難さ *(cont.)*

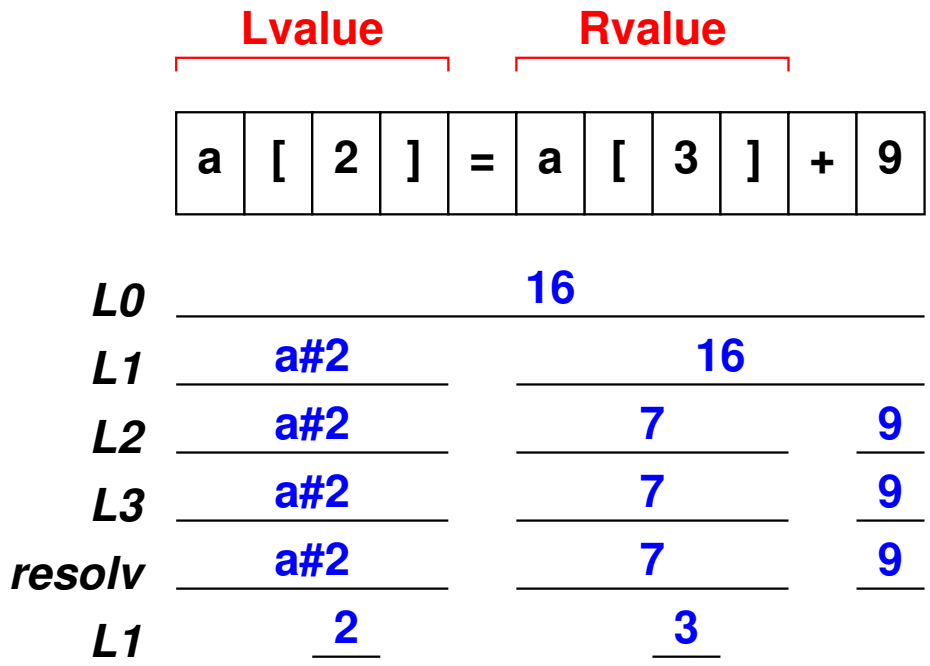
一方、配列では変数名–イコール間に3項目含まれる

```
L0 ::= VAR '[' L1 ']' '=' L1
```

- L0 で L1 呼び出した後 [が出る
  - 代入かどうか判定不能
- 4つ先を読み込む【LL(4)】機構を作るのは大変
- 変数名から]まで、4項目を一括で扱う方法が必要
  - 代入でもそれ以外でも扱えねばならない

今回のアプローチ

- 左辺値を例外扱い
  - 左辺値(イコールの左側)を L1 で扱えるように
  - L1 の結果が変数名であればよい
  - resolv で吸収できる
  - val_t に変数名を格納
- 右辺値は従来通り
  - 変数から値を導出
  - val_t に数や文字列を格納



今回のアプローチ *(cont.)*

- val_t 構造体の内部型に変数を追加

```
#define VTSYM    (3)
```

- メンバは文字列向けの str をそのまま流用
- 作成関数を用意

```
int mkvsym(val_t *dst, char *s);
```

今回のアプローチ *(cont.)*

- 表示ルーチン拡張; $ をつける

```
int
voidp2vstr(char *dst, int dlen, void *src)
{
    val_t *v = (val_t*)src;
    switch(v->type) {
    case VTSYM:
        if(!v->str) { sprintf(dst, "*NULL*"); }
        else { sprintf(dst, "$%s", v->str); }
        break;
...
```

左辺値の判定

- 変数名から、[、式、] が出てきて、次にイコールがあれば左辺値

```
a[3] = 12
```

- こういう代入式(配列要素を複数並べてイコール)は受け付けないので、イコールは確実に左辺値

```
a[3]+b[5] = 82
```

- それ以外は右辺値

左辺値の判定 *(cont.)*

2段がまえにする

1) 変数が出て来ると添字まで評価して要素を確定

```
a[3] → "a#3"
a[3+4] → "a#7"
```

※ # は変数名と添字から辞書のキーを生成する区切り文字

2) 左辺値ならそのまま下位の関数へ返す、右辺値なら辞書から取り出した値を返す

要素を確定する 1) の関数 resolv_var を新設

```
int vsubsep = '#';

int resolv_var(char *vname, int vlen) {
  val_t a1;  int ck;  char sub[BUFSIZ];
  char ctoken[BUFSIZ]; int cttype; int ctop;

  strcpy(ctoken, token);
  cttype = tokentype;  ctop = tokenop;
  ck = gettoken();
  if(cttype==TVAR) {
    if(tokentype==TSYM && tokenop==OLBRK) {
      gettoken();  L1(&a1);
      if(!(tokentype==TSYM && tokenop==ORBRK)) {
```

左辺値の判定 *(cont.)*

```
        fprintf(stderr, "no found expected ’]’\n");
      }
      gettoken();
      voidp2vstr(sub, BUFSIZ, (void*)&a1);
      sprintf(vname,"%s%c%s",ctoken,vsubsep,sub);
    }
    else { strcpy(vname, ctoken);  }
  }
  return 0;
}
```

関数終了時点で、次トークンまで進んでいる点に注意

resolv では 2) を実現

```
int resolv(val_t *rv) {
  val_t a1;  int ck;  vdict_c *pos;
  char vname[BUFSIZ];  int havenext=0;

  switch(tokentype) {
  ...
  case TVAR:
    resolv_var(vname, BUFSIZ);
    havenext = 1;
    mkvsym(rv, vname);
    break;
  ...
  }
```

```
  if(rv->type==VTSYM) {
    if(tokentype==TSYM
      && (tokenop==OEQ || tokenop==OEQPLUS)) {  }
    else {
      pos = vdict_findpos(&vardict, vname);
      if(pos) { *rv = *(val_t*)pos->value;  }
      else {
        mkvnum(rv, 0);
        vdict_add(&vardict, vname,
            (void*)valdup(rv));  }  }
  }
  if(!havenext) {  ck = gettoken();  }
  return 0; }
```

変数の際には gettoken 呼ばない、havenextで切替え

左辺値の判定 *(cont.)*

これにより、以下の配列要素代入式は

```
a[3] = 7
```

次のような変数代入と同等に扱われる

```
a#3 = 7
```

これで、以前と同様、L0 でイコールを比較して代入式を判定できる

※ 右辺値は値を取り出すので特に変わらない

L0

```
int L0(val_t *rv) {
  ...
  L1(&a1);
  if(tokentype==TSYM && tokenop==OEQ) {
    strcpy(vname, a1.str);
    ck = gettoken();
    if(ck<0) {  *rv = a1;  return 0;  }
    ck = L1(&a2);
    pos = vdict_findpos(&vardict, vname);
    if(pos) {
      cv = (val_t*)pos->value;
      if(cv->type==VTSTR) {
```

L0 *(cont.)*

```
      free(cv->str);  cv->str = NULL;
    }
    *cv = a2;
  }
  else {
    vdict_add(&vardict, vname,
        (void*)(valdup(&a2)));
  }
  *rv = a2;
}
...
```

トークン取り出しのポイント

『評価前に必ずトークン取り出しが必要』

- resolv でトークンが数字や文字列なら
  - val_t に数を代入、次のトークンを取り出す
- トークンが変数なら resolv_var へ
  - 次のトークンを取り出す
  - トークンが [なら添字を処理、] まで読み込む
    - トークンを取り出す
  - トークンがイコールなら左辺値扱い

※ 変数の際には resolv での gettoken は不要

トークン取り出しのポイント *(cont.)*

これまでは...

- 各評価関数の最後の gettoken の後はトークン内容を処理しなかった
- 最後の gettoken は次の関数のため

今回は...

- 最後の gettoken の後にトークン内容で条件分岐
- 左辺値の判別のためイコールと比較する
  - 例外的な処理
- 他の関数と処理が重複しないよう注意

まとめ

- 配列は添字付き変数として扱う
- 変数名と添字からキーを作って辞書に格納
- 添字付き変数はイコールが出て来るまで長いので、従来の方法ではパース困難
- 変数の評価を工夫する
  - 変数が左辺値なら名前、右辺値なら値を返す
  - 直後にイコールが出て来る変数は左辺値
- 値格納の構造体で変数名（辞書キー）も格納する

補足: 変数の添字の有無の差

添字なし変数と添字付き変数は別物
変数表を表示すると分かりやすい

```
% ./a.out 'q=3;q[1]=2;q[0]=34;q[2]=7'
(略)
 - - - var-table - - -
3 0 q#1 2
7 1 q 3
8 2 q#0 34
9 3 q#2 7
```

※ハッシュなので登場順でも添字順でもない

補足: 文字列添字

添字に種類の制限がないので、文字列も使える

```
% ./a.out 'a["jaist"]=9;a["jaist"]+3'
9
12
```

※大文字／小文字は区別される

```
% ./a.out 's["foo"]="i";s["bar"]="ro"'
(略)
 - - - var-table - - -
6 0 s#"bar" "ro"
8 1 s#"foo" "i"
```

補足: プログラム規模

ここまでで 1200 行程度

```
% wc ee09.c libvdict-ohash.c libspji.c\
? val2.c val2.h libspji.h libvdict.h
    558    1200   13148 ee09.c
    278     550    5331 libvdict-ohash.c
    210     397    3488 libspji.c
     78     158    1346 val2.c
     17      54     366 val2.h
     20      65     474 libspji.h
     32      88     769 libvdict.h
   1193    2512   24922 total
```

補足: プログラム規模 *(cont.)*

実現した機能

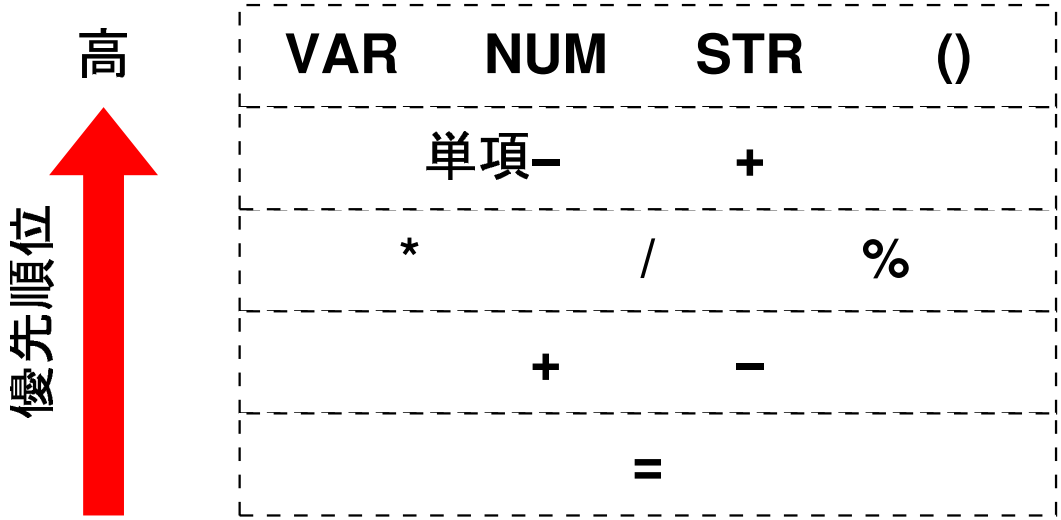
- 数の演算 — 括弧; 乗・除算、剰余; 加・減算
- 文字列の演算 — 連結
- 変数や配列操作 — 添字は文字列でも良い

あると便利な機能

- 数学演算 — 三角関数; べき乗、対数
- 関係演算 — 大小比較
- ユーザ定義関数
- ユーザ定義構造体

おまけ

- ee09.c では単項演算子 + と - に対応した
- 単項演算子+ と - は括弧未満、乗算越

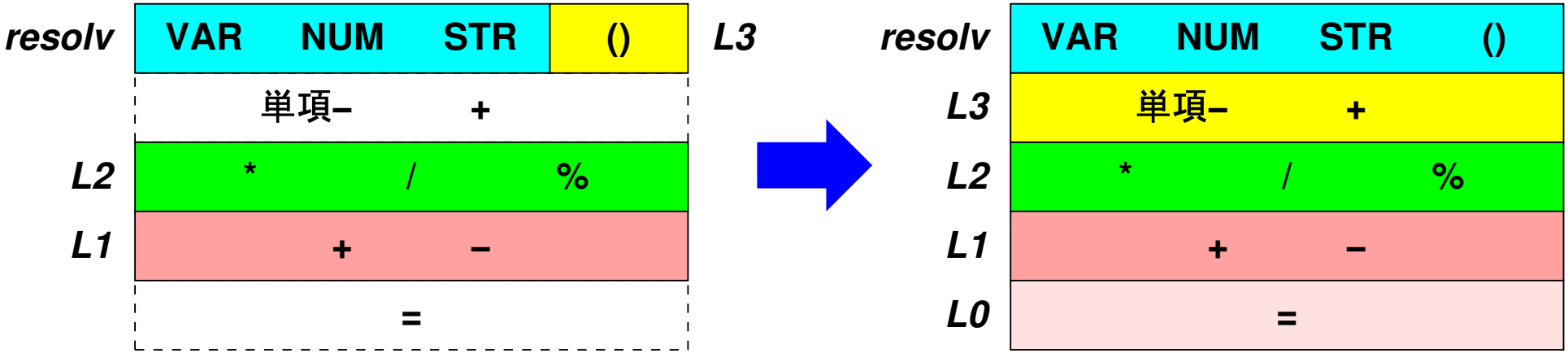


単項演算子と括弧では、括弧の方が優先される

```
-(3+4)
```

おまけ *(cont.)*

- 以前は分かりやすく、括弧向けの L3 を設けた
- 今回（K&R 第二版 A7.2 に合わせる）
  - 括弧を resolv に含める
  - 単項演算向けの L3 を設ける



- 括弧向け L4 を設けてもほぼ等価

演習

1) sin や cos 等の数学関数を扱える数式評価プログラムを作れ★★★
   - 数学関数は標準数学ライブラリを使う
   - ヘッダファイル math.h をインクルード

```
#include <math.h>
```

   - リンク時に -lm を付ける

```
% cc -o a.out foo.o bar.o -lm
```

2) ユーザ定義構造体を扱える数式評価プログラムを作れ★★★

演習 *(cont.)*

3) ユーザ定義関数を扱える数式評価プログラムを作れ★★★
4) 2次元配列を扱えるように先のプログラムを改造せよ★★
5) 4つトークンを読み込んで処理を切替える数式評価プログラムを作れ★★★★