I117（24）数式評価 変数

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

変数

式中に変数を使えると便利

```
a = 32
b = a*7
```

※『split+数式評価』の回でサンプルが出てきた

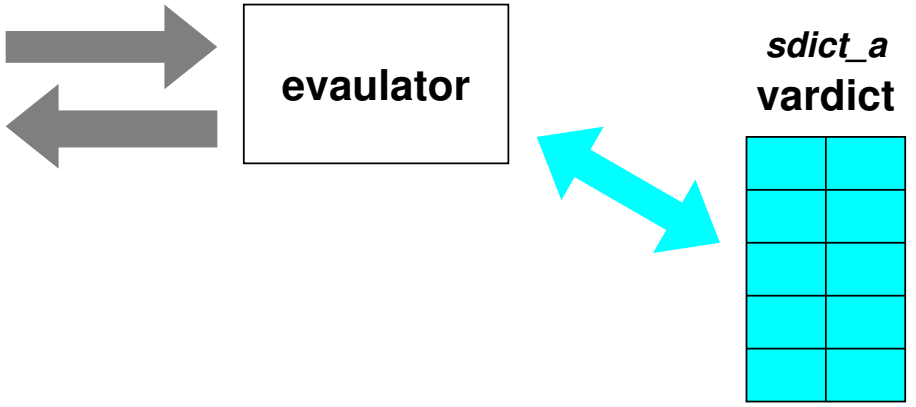
具体的に必要な処理

- 変数名から値への変換
- 変数への値代入
- 格納領域確保(今回は辞書 vardict)

変数格納辞書

辞書は文字列を格納する sdict を用いる

- 将来文字列を格納する可能性を考慮
  - ◇ 数も文字列も格納できるならベスト
  - ◇ どちらか一方なら文字列が有利
- 数は文字列に変換して格納できる
- 初期値を ”0” とする



トークンまわり

トークンで変数名を扱うための種類を新設

```
#define TVAR    (3)
```

代入の演算子はイコールとする

```
#define OEQ     '='
```

変数名はアルファベット列とする

```
        alpha ::= 'a'|'b'|...|'z'
        VAR ::= +alpha
```

gettoken 変更

```
...
else
if(*pos>='a' && *pos<='z') {
    c = 0;
    while(c<BUFSIZ && *pos>='a' && *pos<='z') {
        *q++ = *pos++;  c++;
    }
    *q = '\0';
    tokentype = TVAR;
}
else {
...
```

関数階層

今回の評価の階層を以下のように定める

#	**name**	演算子	摘要
6	resolv		数字→数、変数名→数
5	L3	()	括弧
4	L2	* / %	乗算、除算、剰余
3	L1	+ -	加算、減算
2	L0	=	代入
1	Eeval		式評価

resolv 変更

```
switch(tokentype) {
...
case TVAR:
    pos = sdict_findpos(&vardict, token);
    if(pos) {  *rv = atoi(pos->value);  }
    else {  fprintf(stderr,
                "undefined var-%s\n", token);
    }
    break;
...
```

辞書から値を取り出す

L0 新設

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

注意点

- 変数名が見つかっても代入とは限らない、以下のような記述がありうる

```
a+b
c++
```

- L1 を呼び出した後のイコール登場を待つ

L0 全体の動き

```
if(トークンが変数) {
        変数名を控える
        if(変数が辞書に登録されていない)
                辞書に登録
}
L1（左辺）呼び出し
if(トークンがイコール) {
        L1（右辺）呼び出し
        辞書に登録
}
```

```
int L0(int *rv) {
  int ck, a1, a2, op, lastval;
  sdict_c *pos;
  char vname[BUFSIZ], vval[BUFSIZ];

  vname[0] = '\0';
  if(tokentype==TVAR) {
    strcpy(vname, token);
    pos = sdict_findpos(&vardict, vname);
    if(!pos) { sdict_add(&vardict, vname, "0"); }
  }

  ck = L1(&a1);
  if(ck<0) { return 0; }
```

```
  if(tokentype==TSYM && tokenop==OEQ) {
    ck = gettoken();
    if(ck<0) { return 0; }
    ck = L1(&a2);
    sprintf(vval, "%d", a2);
    pos = sdict_findpos(&vardict, vname);
    if(pos) { free(pos->value);
      pos->value = strdup(vval);
    }
    else { sdict_add(&vardict, vname, vval); }
    *rv = a2;
  }
  return 0;
}
```

L0 新設 *(cont.)*

L0 は Eeval から呼びだされる

```
int Eeval(char *line){
    int v;
    if(line==NULL ||*line=='\0') { return -1; }
    targetline = line;  pos = targetline;
    gettoken();
    L0(&v);
    printf("%d\n", v);
    return 0;
}
```

複数式

一つの式では物足りない、複数の式を評価する

```
% ./a.out 'a=32;b=a*7'
32
224
```

```
% ./a.out 'a=1+2*3;a+=3;(1+2)*3'
7
10
9
```

※これも『split+数式評価』の回で登場

```
int main(int argc, char *argv[]){
    int ck, i;  rec_t exprs;

    sdict_init(&vardict);
    rec_init(&exprs);
    if(argc<=1) {
        printf("usage: <expr>\n"); exit(1); }
    rec_split(&exprs, argv[1], ';');
    for(i=0;i<exprs.used;i++) {
        ck = Eeval(exprs.fields[i]);
        if(ck<0) { break; }
    }
    return 0; }
```

補足 — 文法設計方針

- 混乱を避けるため、演算子の優先順位は何かの言語(例えば C 言語)に準拠することが望ましい
  - ◇ C言語の文法は様々な本に掲載されてる
  - ◇ C言語にない機能は新しい文法を作る
- C言語では変数代入も一つの式

```
a = b = 3;
```

  - ◇ 今回のプログラムでは受け付けきれない
- 先日のフィールド演算子$ は C言語にないので新設した

演習

先のプログラムの変更

1) 変数名にローマ字大文字やアンダースコアを受け付けるように変更せよ★

```
alpha ::= 'a'|'b'|...|'z'|
          'A'|'B'|...|'Z'
VAR ::= +(alpha|'_')
```

演習 *(cont.)*

2) 変数名に数字を受け付けるように変更せよ★
   ただし、先頭はローマ字とせよ

```
alpha ::= 'a'|'b'|...|'z'|
          'A'|'B'|...|'Z'
num ::= '0'|'2'|...|'9'|
VAR ::= (alpha|'_') *(alpha|num|'_')
```

3) 演算子 += に対応せよ★
   - 演算子 = に似せて実現できるだろう

演習 *(cont.)*

4) 演算子 = を続けて記述できるようにせよ★★

```
a = b = 3;
```

5) 演算子 ++ に対応せよ★★★

6) 文字列向けに変更せよ★★★

- 変数で文字列を扱う
- 文字列向けの演算子を新設

7) 数と文字列の両方を扱えるよう拡張せよ★★★★

- 変数で数と文字列の両方を扱う
- 数と文字列の操作が混在した際の挙動を決める