I117（26）数式評価 数・文字列両用

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

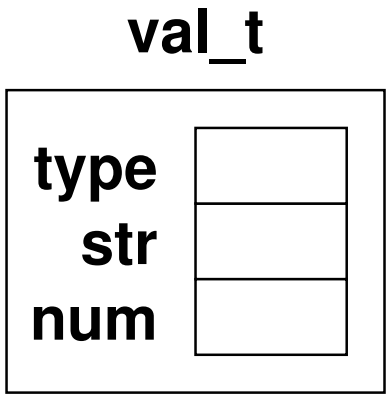
概要

- 数・文字列格納型 val_t の定義と操作関数
- トークン種別追加
- vdict の導入
- 数・文字列の演算の型検討

新規型 **val_t**

数・文字列の両方を格納する
格納している内容の型をメンバ type に記録

```
typedef struct {
    int   type;
    char *str;
    int   num;
} val_t;
#define VTIG    (0)
#define VTSTR   (1)
#define VTNUM   (2)
```



新規型 **val_t** *(cont.)*

新しい型を作る際には、はじめに表示ルーチンを作る

```
int v_print(val_t *v){
    char msg[BUFSIZ];
    voidp2vstr(msg, BUFSIZ, (void*)v);
    printf("%s\n", msg);
}
```

今回に限ってはvdict を利用する都合上、表示用の文字列を生成する関数が必要なので、その関数 void2vstr を流用する

```
int voidp2vstr(char*dst,int dlen,void*src){
  val_t *v = (val_t*)src;
  switch(v->type) {
  case VTSTR:
    if(!v->str) { sprintf(dst, "*NULL*"); }
    else { sprintf(dst, "\"%s\"", v->str); }
    break;
  case VTNUM:
    sprintf(dst, "%d", v->num);  break;
  default:
    sprintf(dst, "*ignore type-%d*", v->type);
    break;
  }
  return 0;  }
```

新規型 **val_t** ***(cont.)***

val_t に数字や文字列を簡単に蓄えるための関数群

```
int mkvnum(val_t *dst, int n) {
    dst->type = VTNUM;  dst->num = n;
    dst->str = NULL; /* dummy */
    return 0;  }
int mkvstr(val_t *dst, char* s) {
    dst->type = VTSTR;  dst->str = strdup(s);
    dst->num = -1234; /* dummy */
    return 0;  }
```

新規型 **val_t** *(cont.)*

strdup に相当する複製関数

```
val_t* valdup(val_t *src) {
    val_t *dst;
    dst = (val_t*)malloc(sizeof(val_t));
    if(dst) {
        memcpy(dst, src, sizeof(val_t));
        if(src->type==VTSTR) {
            dst->str = strdup(src->str); }
    }
    return dst;  }
```

トークン種別追加

gettoken 内にダブルクォートの処理を追加

```
if(*pos==’"’) {  pos++;
    c = 0;
    while(c<BUFSIZ && *pos && *pos!=’"’) {
        *q++ = *pos++;  c++;  }
    *q = ’\0’;
    if(*pos!=’"’)
        fprintf(stderr, "***lack of ’\"’\n");
    pos++;
    tokentype = TSTR;
} ...
```

トークン種別追加 *(cont.)*

文字列向けのトークン種別 TSTR を定義する

```
#define TSTR    (4)
```

vdict 導入

プログラム中の sdict 関連を全て vdict に置換

sdict_a → vdict_a
sdict_c
sdict_findpos → vdict_findpos
sdict_add → vdict_add

val_t と vdict の使い方

- 新規の変数に数 57 を登録

```
mkvnum(&a1, 57);
vdict_add(&vardict, vname, (void*)valdup(&a1));
```

評価関数の結果を val_t に変更

上位関数の結果を受け取る変数の型も val_t へ変更

```
int L0(val_t *rv) {
  val_t a1, a2;
  ...
```

他も同様

```
int resolv(val_t *rv);
int L3(val_t *rv);
int L2(val_t *rv);
int L1(val_t *rv);
```

演算再定義

数と文字列が混在した場合の挙動を決める

```
NUM1 + NUM2 → NUM3      9 + 8 → 17
NUM1 + STR → NUM2       32 + "7colors" → 39
STR1 + NUM → STR2       "abc" + 32 → "abc32"
STR1 + STR2 → STR3      "abc" + "def" → "abcdef"
```

```
NUM1 * NUM2 → NUM3      7 * 4 → 28
STR1 * NUM → STR2       "a" * 3 → "aaa"
```

演算再定義 *(cont.)*

L1 内部

```
case OPLUS:
  if(rv->type==VTNUM) {
    if(a2.type==VTNUM) { rv->num += a2.num; }
    else if(a2.type==VTSTR) { rv->num += atoi(a2.str); }
    else { fprintf(stderr, "***undefined operation\n"); }
  }
  else
  if(rv->type==VTSTR) {
    if(a2.type==VTNUM) {
      sprintf(tmp, "%s%d", rv->str, a2.num);
      free(rv->str);  rv->str = strdup(tmp);
    }
    else if(a2.type==VTSTR) {
```

演算再定義 *(cont.)*

```
      sprintf(tmp, "%s%s", rv->str, a2.str);
      free(rv->str);  rv->str = strdup(tmp);
    }
    else {
      fprintf(stderr, "***undefined operation\n");
    }
  }
  break;
```

演算再定義 *(cont.)*

L2 内部

```
case OTIMES:
  if(rv->type==VTNUM) {
    if(a2.type==VTNUM) { rv->num *= a2.num; }
    else { fprintf(stderr, "***undefined operation\n"); }
  }
  else
  if(rv->type==VTSTR) {
    if(a2.type==VTNUM) {
      tmp[0] = '\0';
      for(i=0;i<a2.num;i++) {
        strcat(tmp, rv->str);
      }
      free(rv->str); rv->str = strdup(tmp);
```

演算再定義 *(cont.)*

```
    }
    else {
      fprintf(stderr, "***undefined operation\n");
    }
  }
  else {
      fprintf(stderr, "***undefined operation\n");
  }
  break;
```

文字列専用演算子

演算子. で文字列連結

```
#define OCAT    '.'
```

gettoken 内部

```
MATCH(OCAT,     ".",    1);
```

L1 内部

```
case OCAT:
  if(rv->type==VTSTR) {
    if(a2.type==VTSTR) {
      sprintf(tmp, "%s%s", rv->str, a2.str);
      free(rv->str); rv->str = strdup(tmp);
    }
    else {
      fprintf(stderr, "***undefined operation\n"); }
  }
  else {
    fprintf(stderr, "***undefined operation\n"); }
  break;
```

実行例

```
% ./ee07 '"abc"+"def"'
"abcdef"
```

```
% ./ee07 'a=3;a+"7colors"'
3
10
```

```
% ./ee07 'a=3;a+="7colors"'
3
10
```

```
% ./ee07 '"abc"*3'
"abcabcabc"
```

実行例 *(cont.)*

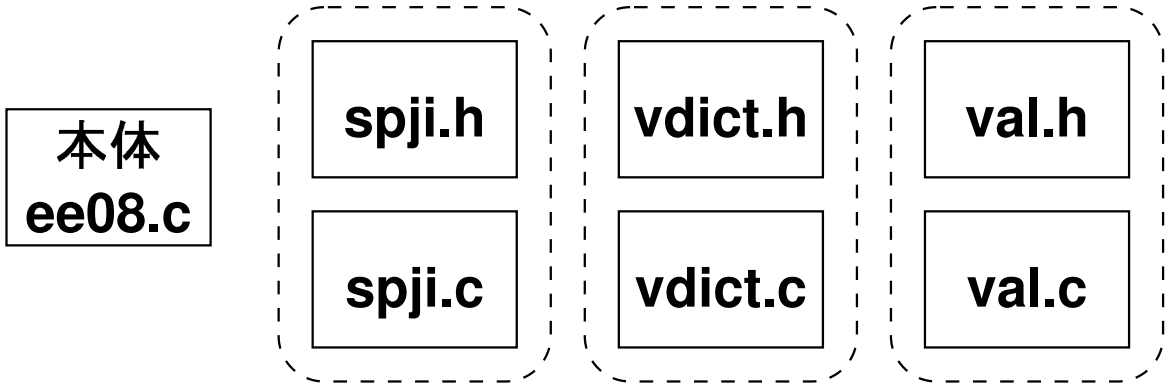
```
% ./ee07 'a="ab";b=a;b+="cd"'
"ab"
"ab"
"abcd"
```

```
% ./ee07 '"abc"."UVW"'
"abcUVW"
```

プログラム構成整理

以下 4 モジュール、7ファイルで構築

- 本体
- val — 値格納データ構造
- spji — split & join, レコード格納データ構造
- vdict — 汎用辞書, 辞書データ構造



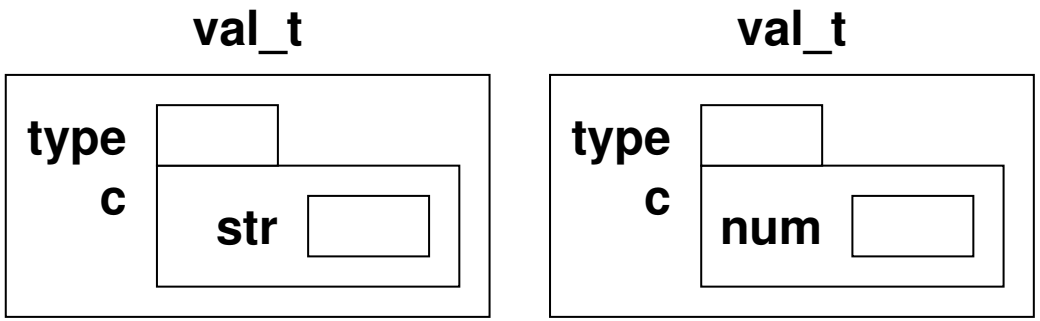
Makefile — タブ文字に注意

```
SRCS=ee08.c libvdict-ohash.c libspji.c val.c
OBJS=${SRCS:.c=.o}
CFLAGS=-DNDEBUG -g

new: ee08
clean:
→$(RM) ee08 $(OBJS)
ee08: $(OBJS)
→$(CC) $(CFLAGS) -o ee08 $(OBJS)

ee08.o: libspji.h libvdict.h val.h
libspji.o: libspji.h
libvdict-ohash.o: libvdict.h
val.o: val.h
```

演習

1) 消費メモリ削減のため、val_t に共用体を導入して先のプログラムを書き直せ★★

```
typedef struct {
  int   type;
  union c {
    char *str;
    int   num;
  }
} val_t;
```



2) 整数だけでなく実数も扱えるように書き直せ★