

I117 (25) void* による辞書

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

内容に依存しない辞書

これまで、内容の型に合わせて辞書を作った

- 文字列向け `sdictionary`
- 数(整数)向け `idictionary`

今後、いろいろな型向けに作るのは手間が多い

- 別の数字向け; `long long` や `double` 等
- 各種ユーザ向け

より汎用向けにするため、内容を `void*` で格納する辞書を作る

void*

qsort や bsearch で登場したように、様々なデータを扱うための型

- 様々な型のデータをポイントできる
- 使う側でキャストが必要

```
char *str = (char*)vp;  
int   num = (int)vp;
```

※ 内容の型は記録していないため、複数の型の混在はできない

void* (cont.)

```
typedef struct {  
    char *key;  
    void *value;  
} vdict_c;  
typedef struct {  
    vdict_c *slot;  
    int      len;  
    int      use;  
} vdict_a;
```

関数インターフェイス

libidict と libsdict の関数定義上の違いは `_add` 関数の値を渡す部分のみ

```
int idict_add(sdect_a*dict,char*xkey,int xvalue);  
int sdect_add(sdect_a*dict,char*xkey,char*xvalue);
```

したがって、libvdict でも値を渡す箇所を変更

```
int vdict_add(vdict_a*dict,char*xkey,void*xvalue);
```

関数インターフェイス (*cont.*)

値の型を知らないため、直接、表示や値に応じた整列はできない

表示は `qsort` のように表示関数を指定する形態に

```
int vdict_print(vdict_a*dict,
                int(*sf)(char *,int,void*));
```

表示関数 `sf` は与えた型 `void*` の値から表示内容を文字列に格納する

引数はそれぞれ、表示内容格納場所(`char*`)と長さ(`int`)、値(`void*`)とする

関数インターフェイス (*cont.*)

libvdict を呼び出す側で関数を用意する
値が文字列の場合

```
int voidp2str(char*dst,int dlen,void*pxval){  
    char *p = (char*)pxval;  
    strcpy(dst, p);  return 0; }
```

値が数(int)の場合

```
int voidp2intstr(char*dst,int dlen,void*pxval){  
    int p = (int)pxval;  
    sprintf(dst, "%d", p);  return 0; }
```

使用例

```
int main() { char line[BUFSIZ];
    vdict_a *sdict; vdict_a *idict;
    idict = vdict_new(); vdict_init(idict);
    sdict = vdict_new(); vdict_init(sdict);
    while(fgets(line, BUFSIZ, stdin)) {
        chomp(line);
        vdict_add(sdict, line, (void*)strdup(line));
        vdict_add(idict, line, (void*)strlen(line));
    }
    vdict_print(sdict, voidp2str);
    vdict_print(idict, voidp2intstr);
}
```

使用例 (*cont.*)

idict と sdict の両方の役割を統合できた
実行結果

```
% ls *. [ch] | ./a.out
4 0 libvdict.h libvdict.h
5 1 libvdict-ohash.c libvdict-ohash.c
6 2 smp.c smp.c
4 0 libvdict.h 10
5 1 libvdict-ohash.c 16
6 2 smp.c 5
```

sdict に文字列が、idict に数が格納できている

ライブラリ内容

表示: 関数を呼び出す以外は、これまでと同様

```
int vdict_print(vdict_a *dict,
               int (*subf)(char *,int,void*)) {
    int i, c=0, ck;
    char tmp[BUFSIZ];
    for(i=0;i<dict->len;i++){
        if(!dict->slot[i].key ||
            !dict->slot[i].key[0]) {
            continue;
        }
        ck = subf(tmp, BUFSIZ, dict->slot[i].value);
        if(ck) { strcpy(tmp, "*error*"); }
    }
}
```

ライブラリ内容 (cont.)

```
        else if(!tmp[0]) { strcpy(tmp, "*empty*"); }
        printf("%d %d %s %s\n",
                i, c, dict->slot[i].key, tmp);
        c++;
    }
    return 0;
}
```

表示される *i* や *c* は、それぞれ、場所(配列添字)と個数を示す

ライブラリ内容 (*cont.*)

追加: idict と同じ

```
int vdict_add(vdict_a*dict, char*xkey, void*xvalue){
    int pos;
    if(!xkey || !xkey[0]) { return -1; }
    if(dict->use>=dict->len) { vdict_expand(dict);
    pos = hash(xkey, dict->len);
    while( dict->slot[pos].key &&
           !dict->slot[pos].key[0]) {
        pos = (pos+1)%dict->len;
    }
}
```

キーがない(NULL)か空(''\0')の場所を探す

ライブラリ内容 (cont.)

```
    if(!dict->slot[pos].key) { dict->use++; }  
    else { free(dict->slot[pos].key); }  
    dict->slot[pos].key = strdup(xkey);  
    dict->slot[pos].value = xvalue;  
    return 0;  
}
```

xvalue の型も内容も分からないため、_add 内部で複製すべきかどうか判断は難しい
ここでは素直に代入し、複製が必要なら呼び出し側で行う

格納とサイズ考察 – 原則

原則はポインタを代入
自動変数をメモリ領域がなくなるので、辞書に蓄える
前にヒープ領域に複製する必要がある

```
struct any x;  
struct any *p = (struct any*)      ←  
    malloc(sizeof(struct any));    ← 正  
*p = x;                             ←  
vdict_add(, , (void*)p);           ←  
  
vdict_add(, , &x);                  ← 誤
```

格納とサイズ考察 – 原則 (*cont.*)

グローバル変数ならメモリ領域がなくなる恐れはないので、心配ない

```
vdict_add(, , &x); ←グローバルなら可
```

読みとる側ではポインタにキャスト

```
int voidp2anystr(char*dst, int dlen, void*pxval){  
    struct any *p = (struct any*)pxval;  
    ...  
}
```

格納とサイズ考察 – 原則 (*cont.*)

文字列はポインタなので同様にキャスト
複製には strdup が便利 <先の sdict の例>

```
vdict_add(, , (void*)strdup("foo-bar"));
```

```
int voidp2str(char*dst,int dlen,void*pxval){  
    char *p = (char*)pxval;
```

格納とサイズ考察 – 例外、直接代入

例外として、void* より小さな型は直接代入も可

<先の idict の例>

```
vdict_add(, , (void*)32);
```

ただし、直接代入した場合は、読みとる方がポインタとして扱わないよう注意が必要

```
int voidp2intstr(char*dst,int dlen,void*pxval){  
    int p = (int)pxval;           ←正  
    int p = *(int*)pxval;         ←誤
```

※ 一般的にユーザ定義型は void* より大きいので、原則のポインタを代入することが多い

long long 辞書

先日、演習で登場した long long の辞書を作る
long long は、void* より大きいのでヒープ領域上に複製して登録する

```
int voidp2lldstr(char*dst,int dlen,void*pxval){  
    long long *p=(long long*)pxval;  
    sprintf(dst, "%lld", *p);    return 0;    }
```

読み出す側では %lld を使うことに注意

long long 辞書 (cont.)

```
int main() {
    vdict_a *lldict;
    char line[BUFSIZ]; long long *tmp;
    lldict = vdict_new(); vdict_init(lldict);
    while(fgets(line, BUFSIZ, stdin)) {
        chomp(line);
        tmp = (long long*)malloc(sizeof(long long));
        *tmp = (long long)strlen(line);
        vdict_add(lldict, line, (void*)tmp);
    }
    vdict_print(lldict, voidp2llstr);
}
```

long long 辞書 (*cont.*)

実行結果

```
% ls *. [ch] | ./a.out
4 0 libvdict.h 10
5 1 libvdict-ohash.c 16
6 2 smp.c 5
7 3 smp11.c 7
```

利用者側だけで lldict も簡単に実現できた

頻度計上

libvdict で頻度計上、第一フィールドを対象にする

```
int main(){
    vdict_a *idict;  vdict_c *p;
    char line[BUFSIZ];  char *key;
    int    tc;  rec_t rec;

    idict = vdict_new(); vdict_init(idict);
    rec_init(&rec);
    while(fgets(line, BUFSIZ, stdin)) {
        chomp(line);
        rec_split(&rec, line, ' ');
    }
}
```

頻度計上 (*cont.*)

```
key = rec.fields[0];
p = vdict_findpos(idict, key);
if(p) { tc = (int)p->value; tc += 1;
        p->value = (void*)tc; }
else { vdict_add(idict, key, (void*)1); }
}
vdict_print(idict, voidp2intstr);
}
```

コンパイルには libspji が必要

頻度計上 (cont.)

```
% ls -l smp* | cat
-rw-r--r-- 1 k-chinen is 1015 Jul 15 14:41 smp.c
-rw-r--r-- 1 k-chinen is 268 Jul 15 12:40 smp.c.bak
-rw-r--r-- 1 k-chinen is 8276 Jul 15 18:41 smp.o
-rwxr-xr-x 1 k-chinen is 22072 Jul 15 18:41 smpfreq
-rw-r--r-- 1 k-chinen is 1053 Jul 15 18:35 smpfreq.c
-rw-r--r-- 1 k-chinen is 18280 Jul 15 18:41 smpfreq.o
-rwxr-xr-x 1 k-chinen is 18096 Jul 15 18:41 smp11
-rw-r--r-- 1 k-chinen is 1690 Jul 15 18:22 smp11.c
-rw-r--r-- 1 k-chinen is 9960 Jul 15 18:41 smp11.o
% ls -l smp* | ./smpfreq
2 0 -rwxr-xr-x 2
7 1 -rw-r--r-- 7
```

頻度計上成功

ヒープ上のカウンタの頻度計上

ヒープ上のデータはポインタで受ける

```
int main() {
    vdict_a *lldict;  vdict_c *p;
    char line[BUFSIZ]; char *key;
    long long *tmp;  rec_t rec;
    lldict = vdict_new(); vdict_init(lldict);
    rec_init(&rec);
    while(fgets(line, BUFSIZ, stdin)) {
        chomp(line);
        rec_split(&rec, line, ' ');
        key = rec.fields[0];
        p = vdict_findpos(lldict, key);
    }
}
```

ヒープ上のカウンタの頻度計上 (*cont.*)

```
    if(p) { tmp = (long long*)p->value;
              *tmp += 1LL;    }
    else { tmp = (long long*)malloc(
              sizeof(long long));
          *tmp = 1LL;
          vdict_add(lldict, key, (void*)tmp); }
}
vdict_print(lldict, voidp2llstr);
}
```

ヒープ上のカウンタの頻度計上 (cont.)

```
% ls -l smp11* | cat
-rwxr-xr-x    1 k-chinen is 18096 Jul 15 18:58 smp11
-rw-r--r--    1 k-chinen is  1690 Jul 15 18:22 smp11.c
-rw-r--r--    1 k-chinen is  9960 Jul 15 18:58 smp11.o
-rwxr-xr-x    1 k-chinen is 22972 Jul 15 18:59 smp11freq
-rw-r--r--    1 k-chinen is  2413 Jul 15 18:59 smp11freq.c
-rw-r--r--    1 k-chinen is 20196 Jul 15 18:59 smp11freq.o
% ls -l smp11* | ./smp11freq
2 0 -rwxr-xr-x 2
7 1 -rw-r--r-- 4
```

演習

- 1) **libvdict** を使って **double** の辞書を扱うプログラムを作れ★
 - **libvdict** 側は変更せず、呼び出し側のみ作れ
- 2) 新しく構造体を設けて、それを値とするプログラムを **libvdict** を使って作れ★★