Recalling…

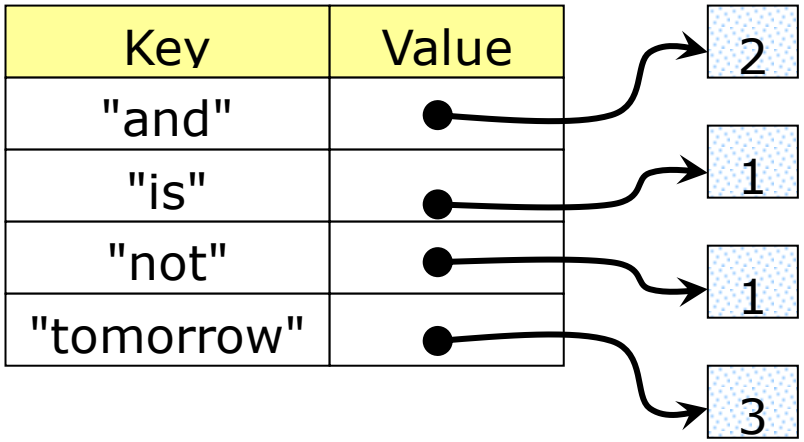
We consider a ‘not-very-useful’ application program that counts how often a word appears in the input, ***<u>one word per line</u>***.

<u>Data:</u>

tomorrow
and
tomorrow
and
tomorrow
is
not

<u>Output:</u>

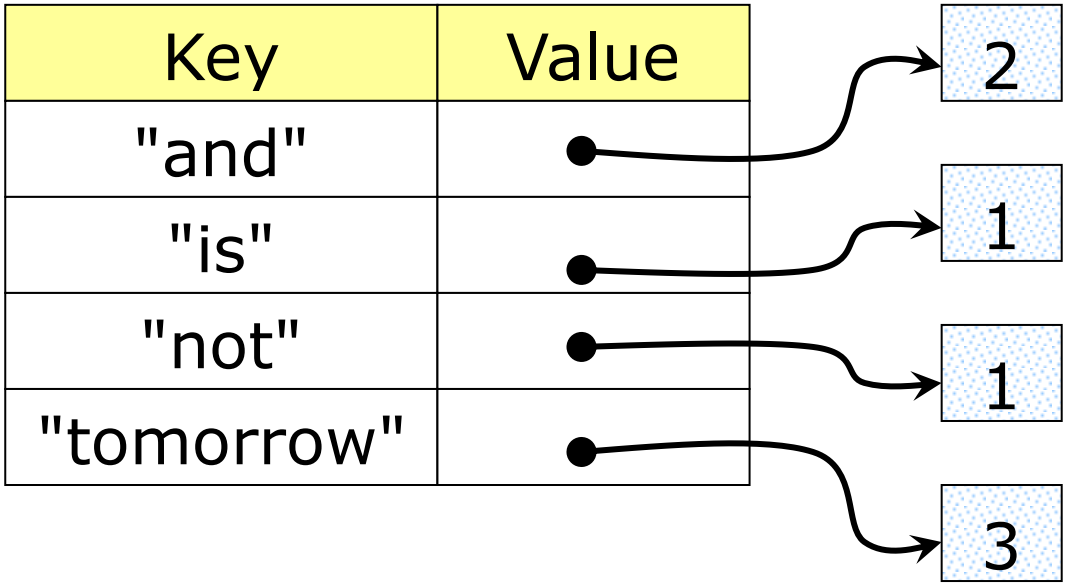
and	2
is	1
not	1
tomorrow	3



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "symtab.h"
typedef struct {int count;} *counterT;
main() {
   char line[80];
   symtabADT table;
   table = EmptySymbolTable();
   scanf("%s", line);
   while (strcmp(line, "***")!=0) {
      RecordWord(table, line); scanf("%s", line);
   }
   DisplayWordFrequencies(table);
}
```

DisplayWordFrequencies(table);

We do not know how to write this function because we do not know what entries are there in the table!



With a table, we can **ONLY**

- Enter an entry;
- Look up an entry

But we ***cannot*** do the following

for (*every entry in the table*) {
***Display the key and the corresponding value*;**
}

Because we do not know what entries there are in the table.

What Should We Do?

What Should We Do?

SOLUTION: To add one more operation for the symbol table ADT. That is, we add it to both **symtab.h** and **symtab.c**.

This operation is

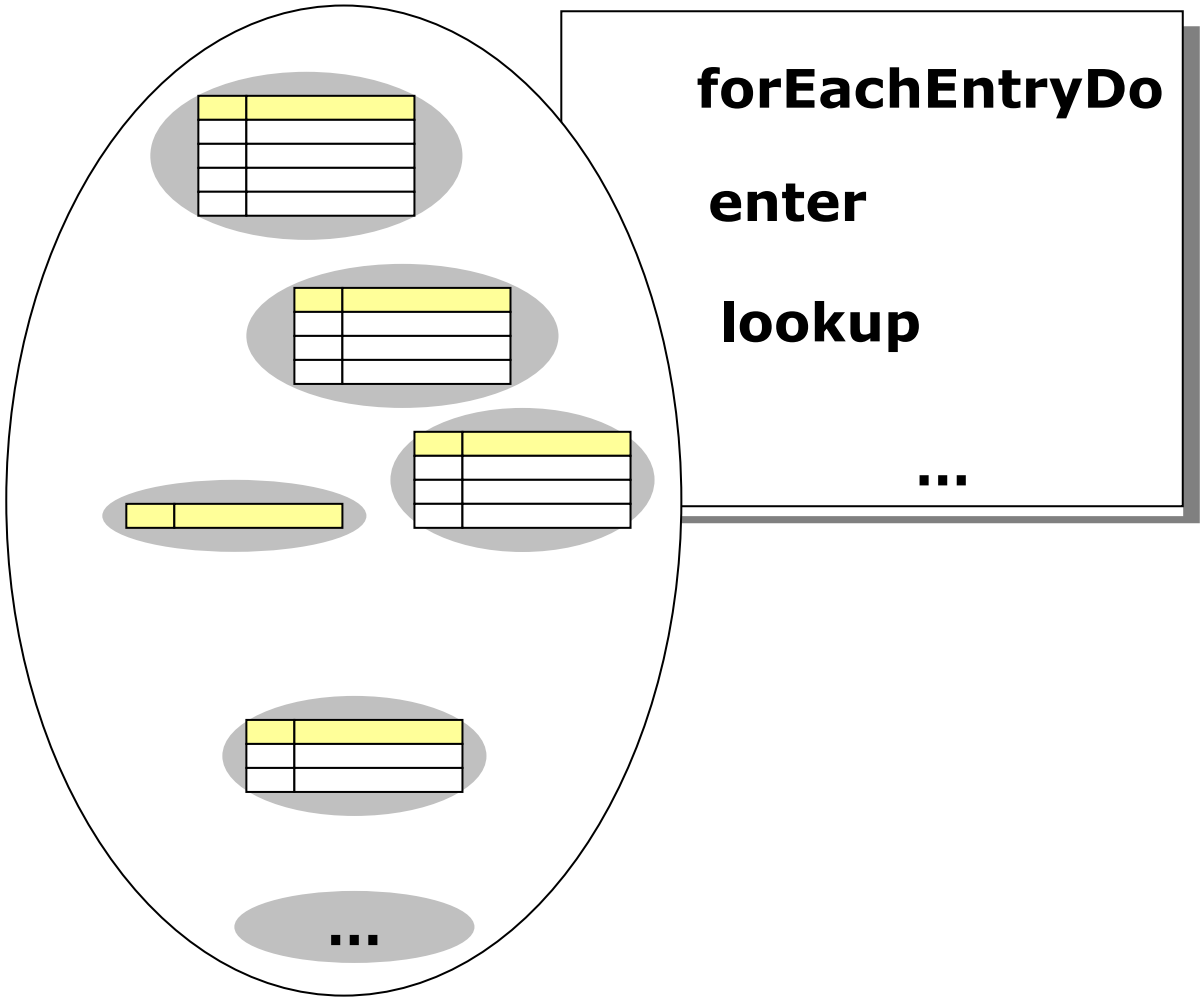
forEachEntryDo(f, table)

where **f(key,value)** is a function accepting a **key** and a **value**.

The operation **forEachEntryDo** performs the following:

```
for (every entry (key,value) in the table) {
   Call f(key,value);
}
```

Symbol Table ADT



Example.

If **table** contains

$$<k_1, v_1>, <k_2, v_2>, \ldots, <k_n, v_n>$$

Then the call **forEachEntryDo(f, table)** effectively performs the following *n* statements in sequence:

$$\mathbf{f}(k_1, v_1);$$
$$\mathbf{f}(k_2, v_2);$$
$$\ldots$$
$$\mathbf{f}(k_n, v_n);$$

f(k, v);

Therefore, type of **f** is

```
void f(char*, void*);
```

A function pointer **fp** that points to **f** can be declared as

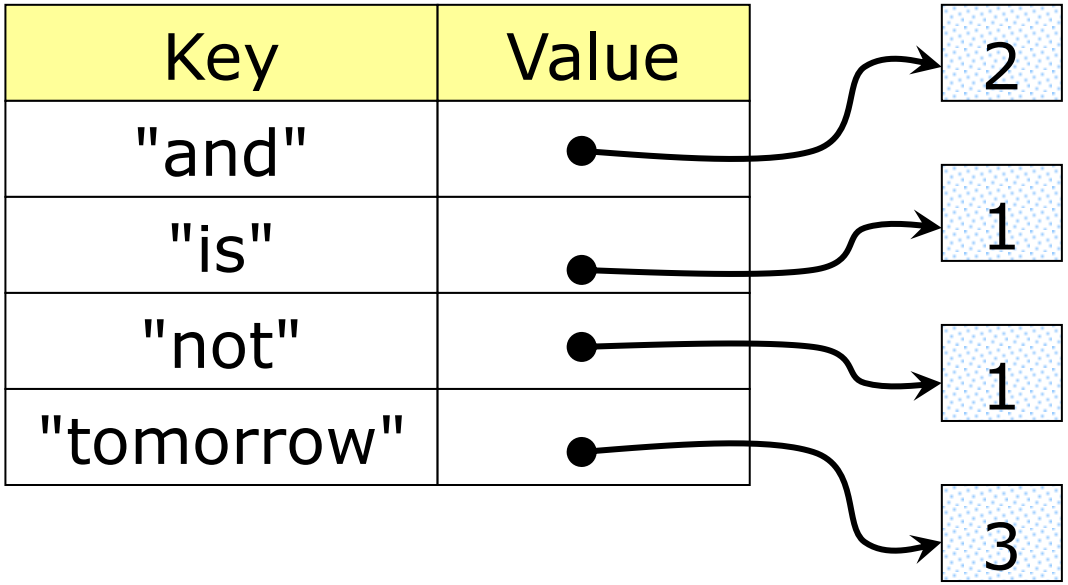
```
void (*fp)(char*, void*);
```

Alternatively (and equivalently)

```
typedef void (*symtabFnT)(char*, void*);
symtabFnT fp;
```

Example.

```
void DisplayEntry(char *key, void *c) {
   printf("%s\t%d\n", key, ((counterT) c)->count);
}
```

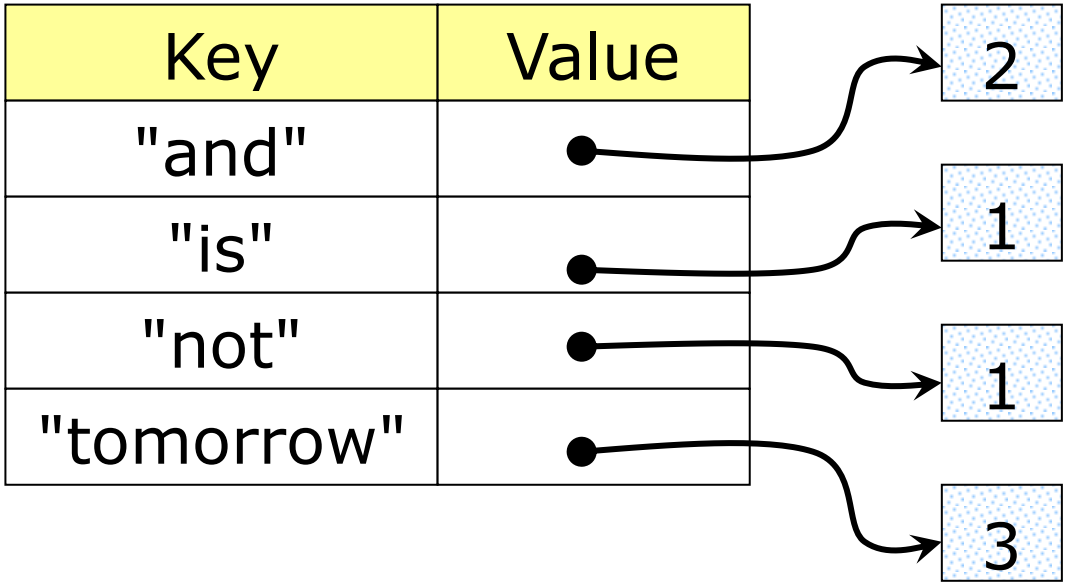


Then

forEachEntryDo(DisplayEntry, table)

would cause the following to be printed:

and	**2**
is	**1**
not	**1**
tomorrow	**3**



Prototype of **forEachEntryDo**:

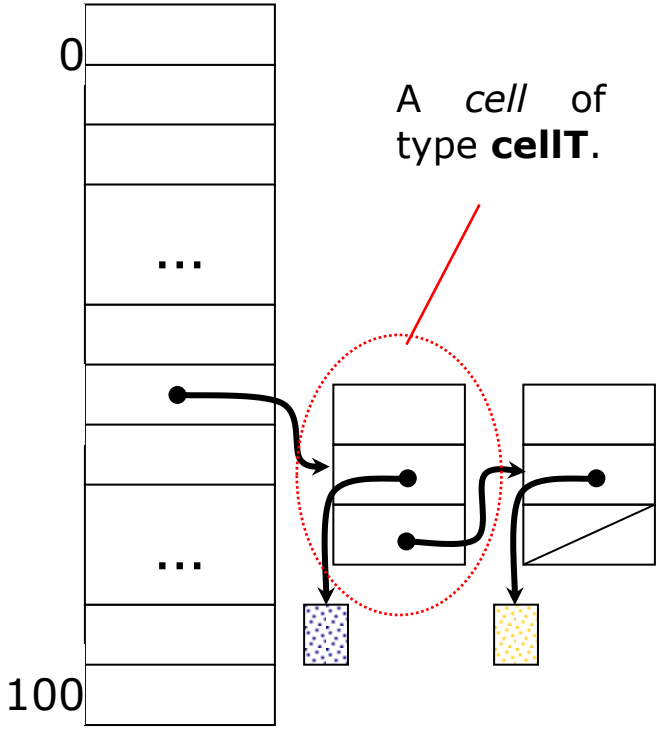
```
void forEachEntryDo(symtabFnT, symtabADT);
```

where

```
typedef void (*symtabFnT)(char*, void*);
```

How do we write this function **forEachEntryDo**?

```
void forEachEntryDo(symtabFnT f, symtabADT table) {

   int i; cellT *cp;

   for (i=0; i<101; i++)
      for (   cp=table->buckets[i];
              cp!=NULL;
              cp=cp->next)
         f(cp->key, cp->value);
}
```



Isn't this <u>*very*</u> simple?

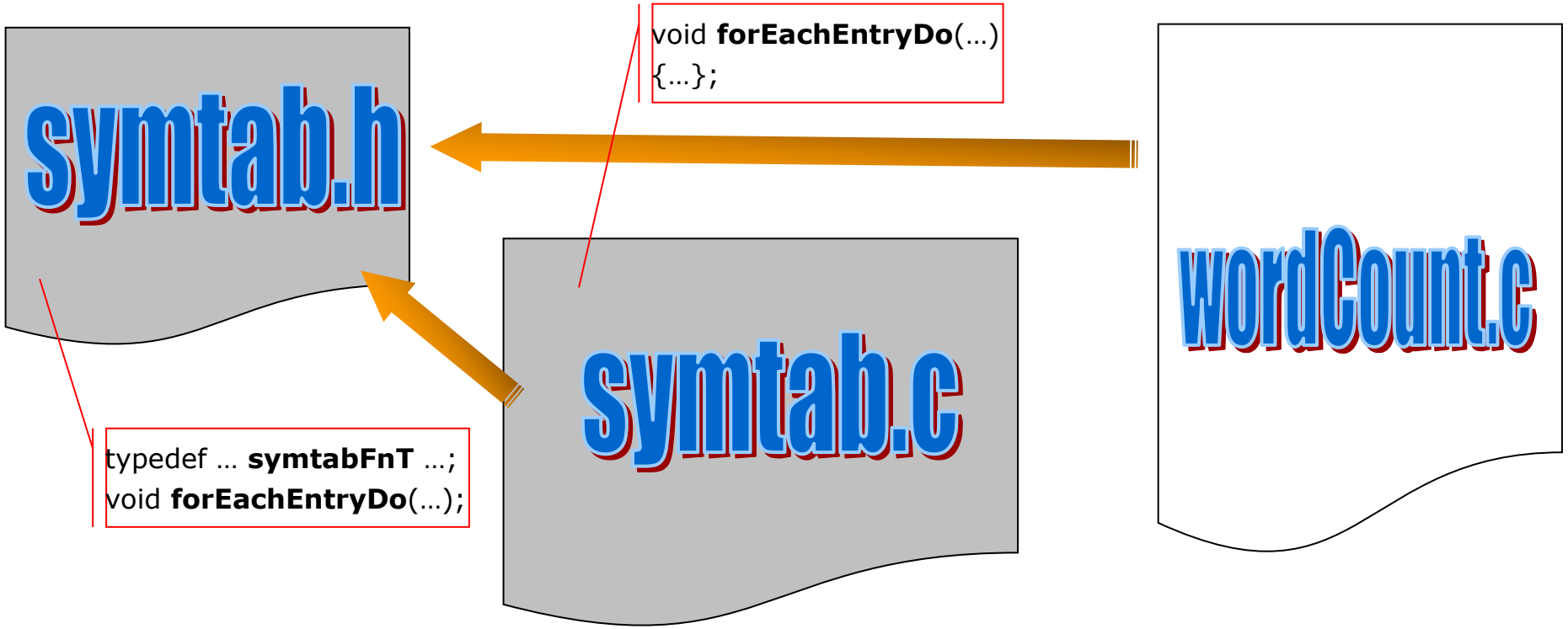
In summary, we need to do the following:

To add to **symtab.h**:

```
typedef void (*symtabFnT)(char*, void*);
void forEachEntryDo(symtabFnT, symtabADT);
```

To add to **symtab.c**:

```
void forEachEntryDo(symtabFnT f, symtabADT table) {
  int i; cellT *cp;
  for (i=0; i<101; i++)
    for (cp=table->buckets[i]; cp!=NULL; cp=cp->next)
      f(cp->key, cp->value);
}
```



The last step is now straightforward:

```
void DisplayWordFrequencies(symtabADT table) {
  forEachEntryDo(DisplayEntry, table);
}
```

```
/* wordCount.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "symtab.h"
typedef struct {int count;} *counterT;
main() {
   char line[80];  symtabADT table;
   table = EmptySymbolTable();
   scanf("%s", line);
   while (strcmp(line, "***")!=0) {
      RecordWord(table, line); scanf("%s", line);
   }
   DisplayWordFrequencies(table);
}
```

Finally, in **wordCount.c**:

```
void DisplayEntry(char *key, counterT c) {
  printf("%s\t%d\n", key, ((counterT) c)->count);
}

void DisplayWordFrequencies(symtabADT table) {
  printf("Word Frequency Table:\n");
  forEachEntryDo(DisplayEntry, table);
  printf("End of Word Frequency Table.\n");
}
```

Terminologies:

Here **forEachEntryDo** is an example of **Mapping Functions** and

DisplayEntry is an example of **callback functions**.

Command Dispatch Tables

Now we look at another simple application of symbol tables.

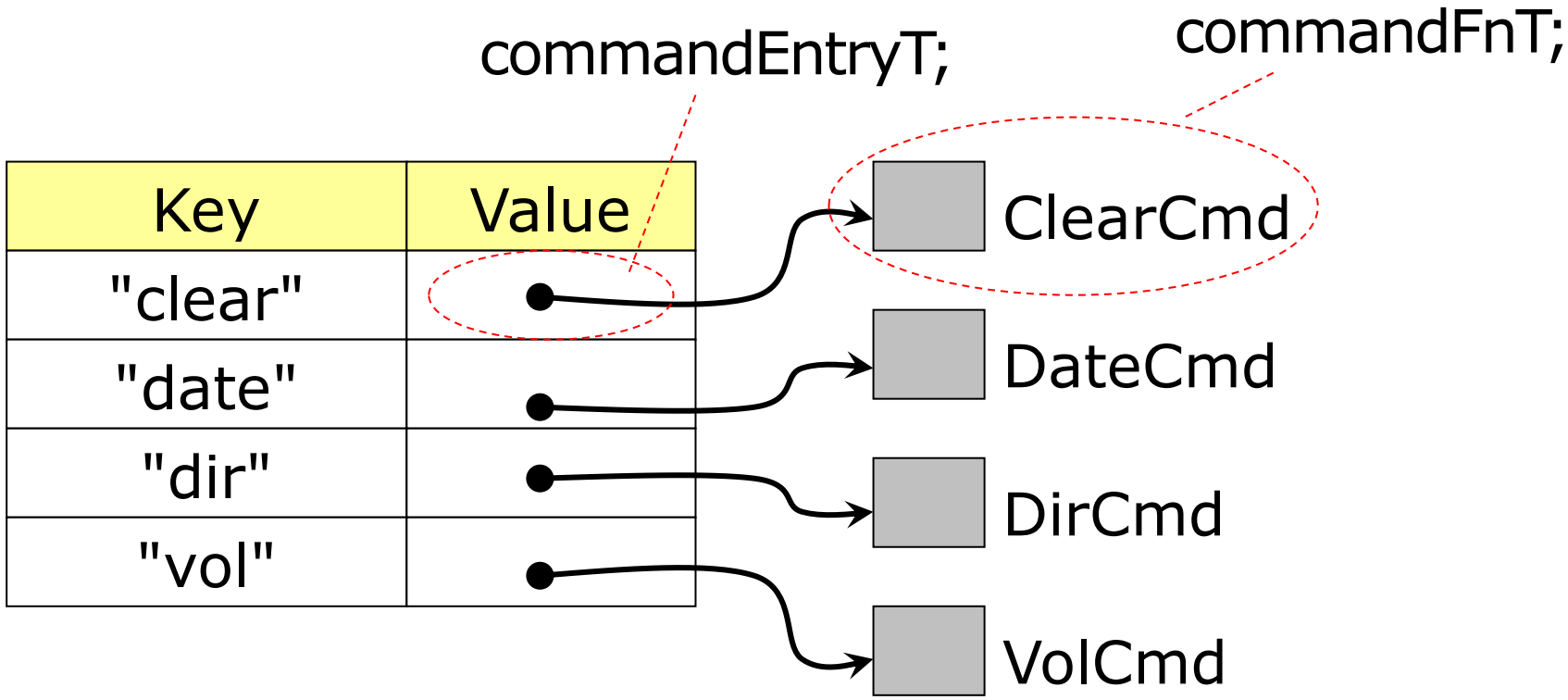
```
D:\>verify
VERIFY is off.

D:\>vol
 Volume in drive D is DATA
 Volume Serial Number is 15E8-3B29

D:\>dir/w
 Volume in drive D is DATA
 Volume Serial Number is 15E8-3B29

 Directory of D:\

[lhf]            [My Documents]   [PQDI]           [Ghost]
[Unknown Artist] [My Music]       stub.log         [I386]
               1 File(s)            554 bytes
               7 Dir(s)   1,708,384,256 bytes free

D:\>date
The current date is:
Enter the new date: (dd-mm-yy)

D:\>
D:\>
```

Consider the following function (that contains a **skip chain**):

```
void ExecuteCommand(char* cmd) {
   if (strcmp(cmd, "clear")==0)
      ClearCmd();
   else if (strcmp(cmd, "date")==0)
      DateCmd();
   else if (strcmp(cmd, "dir")==0)
      DirCmd();
   else if (strcmp(cmd, "vol")==0)
      VolCmd();
}
```

Another approach is to use a symbol table.

```
typedef void (*commandFnT)(void);
typedef struct {commandFnT fn;} *commandEntryT;
```

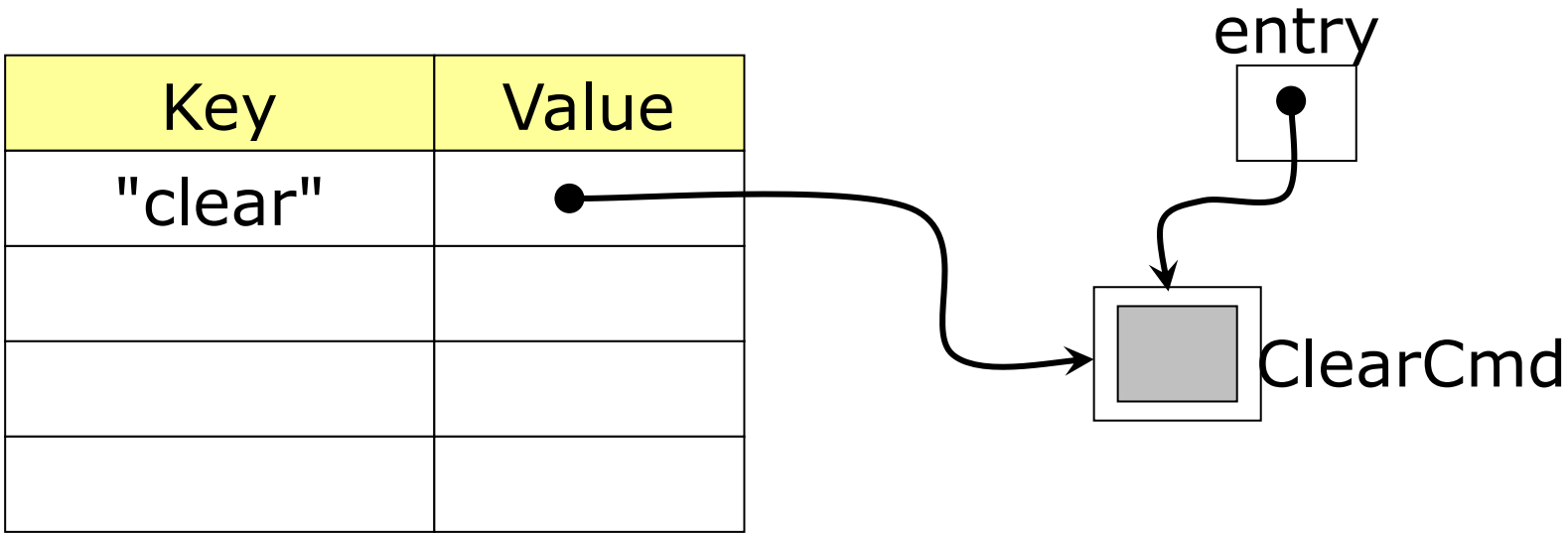


```
typedef void (*commandFnT)(void);
typedef struct {commandFnT fn;} *commandEntryT;

symtabADT commandTable;

void InitCommandTable(void) {
  commandTable = EmptySymbolTable();
  DefineCommand("clear", ClearCmd);
  DefineCommand("date", DateCmd);
  DefineCommand("dir", DirCmd);
  DefineCommand("vol", VolCmd);
}
```

```
void DefineCommand(char* cmd, commandFnT fn) {

  commandEntryT entry;

  entry = (commandEntryT) malloc(sizeof(*entry));
  entry->fn = fn;
  Enter(commandTable, cmd, entry);
}
```

Note

- **entry = (commandEntryT) malloc(sizeof(*entry));**
- **entry->fn = fn;**
- **Enter(commandTable, cmd, entry);**



```
void ExecuteCommand(char* cmd) {

  commandEntryT entry;

  entry = (commandEntryT) Lookup(commandTable, cmd);
  if (entry == NULL)
    printf("Undefined command!\n");
  else
    (*(entry->fn))();
    /* Or simply entry->fn(); */
}
```