Header file: mydefs.h
or better dictionary.h

```
#ifndef DICTIONARY_H
#define DICTIONARY_H

typedef int boolean;
#define FALSE 0
#define TRUE 1
typedef struct n {
    char *key, *value;
    boolean mycle;
    struct n *next;
} NODEREC, *NODE;

#define HASH_SIZE 101        could be in .c file
// sets key to value in the dictionary
boolean set(char *key, char *value);
// retrieves value associated with key or NULL if none
char *get(char *key);
void initialize(); // sets all entries to NULL in table
#endif
```

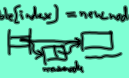
dictionary.c:

```
#include "dictionary.h"
NODE thetable[HASH_SIZE];
// returns hash value of a string
static int hash(char *key)
{
    int ret;
    for(ret=0; *key; key++)
        ret = (37*ret + *key) % HASH_SIZE;
    return ret;
}
```

. . .

insert: compute hash value
- search for the record in the list at that value
- if not there, allocate a new NODE, insert it


```
// insert node at the beginning of a list
int index = hash(key);
// search thetable[index] for the key
// assume key not found
NODE newnode = (NODE)malloc(
                    sizeof(NODEREC));
newnode->key = key;
newnode->value = value;
// deal with mycle stuff
newnode->next = thetable[index];
thetable[index] = newnode;
```



Searching for a key in a linked list:

```
NODE p;
for(p = thetable[index];
    p != NULL;
    p = p->next)
  if (!strcmp(p->key, key))
    break;
if (p != NULL)
  // found it!
```

---

flex internals — theory

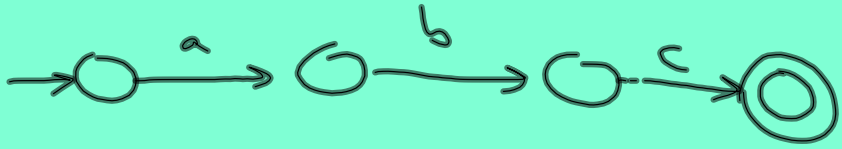
regexp → NFA (nondeterministic finite automaton) → DFA (deterministic finite automaton) → minimize → minimal DFA

output C code simulating this DFA (one for each regexp)

An NFA is a directed graph, where
- vertices are called states
- edges are transitions
- start state
- each state is either accepting (final) or rejecting (nonfinal)
- each transition has a label: a set of chars

Use ε to denote the empty string (""), length 0.

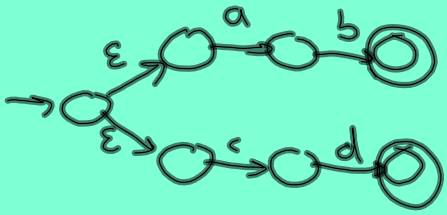
"abc"



"→○" denotes start state

"◎" denotes accepting state

ab|cd



An NFA N accepts a string w if starting in the start state, there is a way to traverse N while reading the input so that

1. each symbol read matches the label of the edge traversed
2. ε-edges can be traversed without reading a char
3. end up in an accepting state after reading the entire input (left to right).

Jan 24-12:05 PM