Word Recognizer, verb/not a verb

```
%{
/*
 * this sample demonstrates (very) simple recognition:
 * a verb/not a verb.
 */

%}
%%

[\t ]+      /* ignore white space */ ;

is |
am |
are |
were |
was |
be |
being |
been |
do |
does |
did |
will |
would |
should |
can |
could |
has |
have |
had |
go      { printf("%s: is a verb\n", yytext); }

[a-zA-Z]+   { printf("%s: is not a verb\n", yytext); }

.|\n        { ECHO; /* normal default anyway */ }
%%

main()
{
    yylex();
}
```

Word Recognizer, verbs and other parts of speech

```
%{
/*
 * We expand upon the first example by adding recognition of some other
 * parts of speech.
 */

%}
%%

[\t ]+      /* ignore white space */ ;

is |
am |
are |
were |
was |
be |
being |
been |
do |
does |
did |
will |
would |
should |
can |
could |
has |
have |
had |
go      { printf("%s: is a verb\n", yytext); }

very |
simply |
gently |
quietly |
calmly |
angrily     { printf("%s: is an adverb\n", yytext); }

to |
from |
behind |
above |
below |
between |
below       { printf("%s: is a preposition\n", yytext); }

if |
then |
and |
but |
or      { printf("%s: is a conjunction\n", yytext); }

their |
my |
your |
his |
her |
its     { printf("%s: is an adjective\n", yytext); }

I |
you |
he |
she |
we |
they        { printf("%s: is a pronoun\n", yytext); }


[a-zA-Z]+ {
     printf("%s:  don't recognize, might be a noun\n", yytext);
    }

\&.|\n      { ECHO; /* normal default anyway */ }

%%

main()
{
    yylex();
}
```

Word Recognizer with a Symbol Table, Definition Section

```
%{
/*
 * Word recognizer with a symbol table.
 */

enum {
    LOOKUP = 0, /* default - looking rather than defining. */
    VERB,
    ADJ,
    ADV,
    NOUN,
    PREP,
    PRON,
    CONJ
};

int state;

int add_word(int type, char *word);
int lookup_word(char *word);
%}
```

Word Recognizer with a Symbol Table, Rules Section

```
%%
\n  { state = LOOKUP; } /* end of line, return to default state */

^verb   { state = VERB; }
^adj    { state = ADJ; }
^adv    { state = ADV; }
^noun   { state = NOUN; }
^prep   { state = PREP; }
^pron   { state = PRON; }
^conj   { state = CONJ; }

[a-zA-Z]+  {
        /* a normal word, define it or look it up */
         if(state != LOOKUP) {
            /* define the current word */
            add_word(state, yytext);
         } else {
        switch(lookup_word(yytext)) {
        case VERB: printf("%s: verb\n", yytext); break;
        case ADJ: printf("%s: adjective\n", yytext); break;
        case ADV: printf("%s: adverb\n", yytext); break;
        case NOUN: printf("%s: noun\n", yytext); break;
        case PREP: printf("%s: preposition\n", yytext); break;
        case PRON: printf("%s: pronoun\n", yytext); break;
        case CONJ: printf("%s: conjunction\n", yytext); break;
        default:
            printf("%s:  don't recognize\n", yytext);
            break;
        }
            }
          }

.   /* ignore anything else */ ;

%%
```

Word Recognizer with a Symbol Table, User Subroutines Section

```
main()
{
    yylex();
}
/* define a linked list of words and types */
struct word {
    char *word_name;
    int word_type;
    struct word *next;
};

struct word *word_list; /* first element in word list */

extern void *malloc();

int add_word(int type, char *word)
{
    struct word *wp;
    if(lookup_word(word) != LOOKUP) {
        printf("!!! warning: word %s already defined \n", word);
        return 0;
    }

    /* word not there, allocate a new entry and link it on the list */
    wp = (struct word *) malloc(sizeof(struct word));

    wp->next = word_list;
    /* have to copy the word itself as well */

    wp->word_name = (char *) malloc(strlen(word)+1);
    strcpy(wp->word_name, word);
    wp->word_type = type;
    word_list = wp;
    return 1;   /* it worked */
}

int lookup_word(char *word)
{
    struct word *wp = word_list;
    /* search down the list looking for the word */
    for(; wp; wp = wp->next) {
        if(strcmp(wp->word_name, word) == 0)
            return wp->word_type;
    }
       return LOOKUP;  /* not found */
}
```

Build a lexical analyzer to be used by a higher-level parser

```
%{
/* We now build a lexical analyzer to be used by a higher-level parser. */
#include "y.tab.h"  /* token codes from the parser */
#define LOOKUP 0    /* default - not a defined word type. */
int state;
%}

%%
\n  { state = LOOKUP; }
\.\n    {   state = LOOKUP;
          return 0; /* end of sentence */  }
^verb   { state = VERB; }
^adj    { state = ADJECTIVE; }
^adv    { state = ADVERB; }
^noun   { state = NOUN; }
^prep   { state = PREPOSITION; }
^pron   { state = PRONOUN; }
^conj   { state = CONJUNCTION; }

[a-zA-Z]+ {
         if(state != LOOKUP) {
            add_word(state, yytext);
         } else {
        switch(lookup_word(yytext)) {
        case VERB:
          return(VERB);
        case ADJECTIVE:
          return(ADJECTIVE);
        case ADVERB:
          return(ADVERB);
        case NOUN:
          return(NOUN);
        case PREPOSITION:
          return(PREPOSITION);
        case PRONOUN:
          return(PRONOUN);
        case CONJUNCTION:
          return(CONJUNCTION);
        default:
          printf("%s:  don't recognize\n", yytext);
          /* don't return, just ignore it */
        }
            }
          }
.   ;
%%
```

same User Subroutines Section

```
/* define a linked list of words and types */
struct word {
    char *word_name;
    int word_type;
    struct word *next;
};

struct word *word_list; /* first element in word list */

extern void *malloc();

int add_word(int type, char *word)
{
    struct word *wp;

    if(lookup_word(word) != LOOKUP) {
        printf("!!! warning: word %s already defined \n", word);
        return 0;
    }

    /* word not there, allocate a new entry and link it on the list */

    wp = (struct word *) malloc(sizeof(struct word));

    wp->next = word_list;

    /* have to copy the word itself as well */

    wp->word_name = (char *) malloc(strlen(word)+1);
    strcpy(wp->word_name, word);
    wp->word_type = type;
    word_list = wp;
    return 1;   /* it worked */
}

int lookup_word(char *word)
{
    struct word *wp = word_list;

    /* search down the list looking for the word */
    for(; wp; wp = wp->next) {
        if(strcmp(wp->word_name, word) == 0)
            return wp->word_type;
    }
    return LOOKUP;  /* not found */
}
```

Simple YACC sentence parser

```
%{
/*
 * A lexer for the basic grammar to use for recognizing english sentences.
 */
#include <stdio.h>
%}

%token NOUN PRONOUN VERB ADVERB ADJECTIVE PREPOSITION CONJUNCTION

%%
sentence: subject VERB object   { printf("Sentence is valid.\n"); }
    ;

subject:    NOUN
    |   PRONOUN
    ;

object:     NOUN
    ;
%%

extern FILE *yyin;

main()
{
    while(!feof(yyin)) {
        yyparse();
    }
}

yyerror(s)
char *s;
{
    fprintf(stderr, "%s\n", s);
}
```