# **Assignment 3**

# **Aim :**

Write a program to implement a lexical analyzer in LEX tool

3.a -- For parts of speech for subset of ENGLISH language without using SYMBOL TABLE

3.b -- For parts of speech for subset of ENGLISH language with SYMBOL TABLE

3.c -- Without using SYMBOL TABLE for subset of ‘C’ programming language

3.d -- With SYMBOL TABLE for subset of ‘C’ programming language

**Code : 3a**

%option noyywrap

%{

%}

%%

[\t ]+ ;

is |

are |

was |

am |

done |

sing |

dances |

plays |

was |

were |

be |

been |

should |

could |

would |

holds |

proved |

say {printf("%s -> VERB\n", yytext);}

good |

great |

bad |

worst |

very |

sharply |

gently |

quietly |

calmly |

angrily {printf("%s -> ADVERB\n", yytext);}

to |

from |

behind |

below |

between |

below |

above {printf("%s -> PREPOSITION\n", yytext);}

and |

or |

but |

since |

hence |

then |

if {printf("%s -> CONJUNCTION\n", yytext);}

I |

he |

she |

it |

they |

me |

we {printf("%s -> PRONOUN\n", yytext);}

their |

my |

Your |

his |

her |

its {printf("%s -> ADJECTIVE\n", yytext);}

a |

an |

The {printf("%s -> ARTICLE\n", yytext);}

[a-zA-Z]+ {printf("%s -> can be a NOUN\n", yytext);}

.|\n {ECHO;}

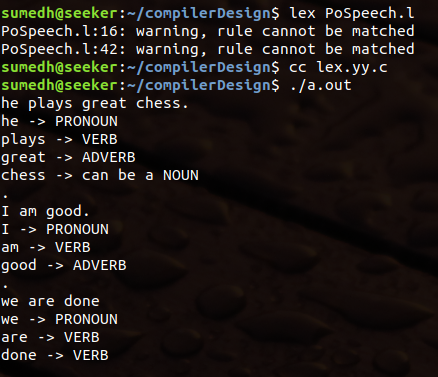
%%

void main(){

yylex();

}

**Output: 3a**



**Code : 3b**

%option noyywrap

%{

#include <stdio.h>

#include <string.h>

enum type { UNKNOWN = 0, NOUN, VERB, ADV, ADJ, PRON, CONJ, PREP};

enum type state = UNKNOWN;

void add\_word(char\*, int);

int find\_word\_type(char\*);

void print\_list();

%}

%%

\n {state = UNKNOWN;}

^noun {state = NOUN;}

^verb {state = VERB;}

^adv {state = ADV;}

^adj {state = ADJ;}

^pro {state = PRON;}

^conj {state = CONJ;}

^prep {state = PREP;}

[a-zA-Z]+ {

if(state != UNKNOWN) {

add\_word(yytext, state);

}

else {

switch(find\_word\_type(yytext)) {

case NOUN : printf("%s\t -> NOUN\n", yytext); break;

case VERB : printf("%s\t -> VERB\n", yytext); break;

case ADV : printf("%s\t -> ADVERB\n", yytext); break;

case ADJ : printf("%s\t -> ADJECTIVE\n", yytext); break;

case PRON : printf("%s\t -> PRONOUN\n", yytext); break;

case CONJ : printf("%s\t -> CONJUNCTION\n", yytext); break;

case PREP : printf("%s\t -> PREPOSITION\n", yytext); break;

default : printf("%s\t -> NOT IN DICTIONARY\n", yytext);

}

}

}

. ;

%%

void main() {

yylex();

}

struct word {

char\* name;

int type;

struct word \* next;

};

struct word \*word\_list;

extern void \*malloc();

void add\_word(char\* word, int state) {

if(word\_list && find\_word\_type(word) != UNKNOWN) {

printf("WARNING : The word '%s' is already defined.\n", word);

return;

}

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

wp->name = (char \* ) malloc(strlen(word) + 1);

strcpy(wp->name, word);

wp->type = state;

wp->next = word\_list;

word\_list = wp;

//print\_list();

}

int find\_word\_type(char\* word) {

struct word \*wp = word\_list;

for(; wp; wp=wp->next) {

if(strcmp(wp->name, word) == 0)

return wp->type;

}

return UNKNOWN;

}

void print\_list() {

struct word \*tmp = word\_list;

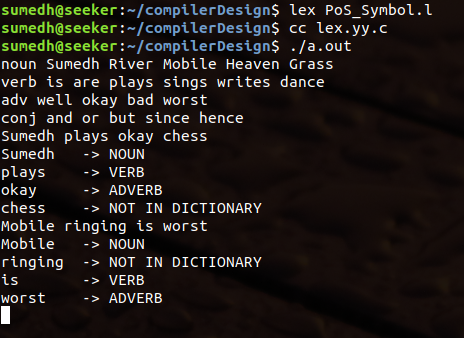
for(;tmp; tmp=tmp->next)

printf("(%s -> %d) -> ", tmp->name, tmp->type);

printf("NULL\n");

}

**Output: 3b**



**Code : 3c**

%option noyywrap

%{

%}

%%

[\t ]+ ;

int |

char |

float |

double |

struct |

union {printf("%s : data type\n", yytext);}

[a-zA-Z][a-zA-Z0-9\_]\* {printf("%s : variable/identifier\n", yytext);}

[0-9]+ {printf("%s : constant/value\n", yytext);}

[\*+-/] {printf("%s : arithmatic operator\n", yytext);}

= {printf("%s : assignment operator\n", yytext);}

\< |

\> |

== |

\<= |

\>= {printf("%s : comparision operator\n", yytext);}

; {printf("%s : delimiter\n", yytext);}

.|\n {ECHO;}

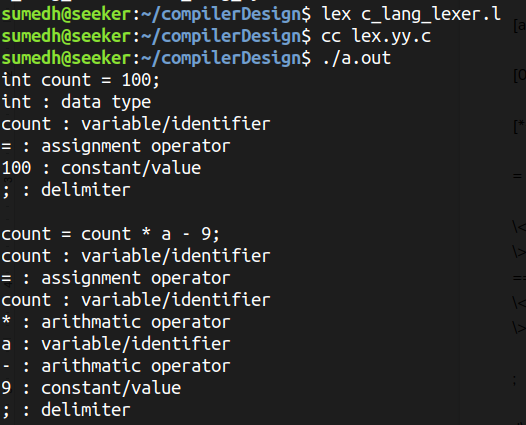
%%

void main() {

yylex();

}

**Output: 3c**



**Code : 3d**

%option noyywrap

%{

#include <stdio.h>

#include <string.h>

enum type { UNKNOWN = 0, KEYWORD, VAR, CONST, OPERATOR, DELIM, SPECIALCHAR};

enum type state = UNKNOWN;

void add\_word(char\*, int);

int find\_word\_type(char\*);

void print\_list();

%}

%%

\n {state = UNKNOWN;}

^keyword {state = KEYWORD;}

^var {state = VAR;}

^const {state = CONST;}

^operator {state = OPERATOR;}

^delim {state = DELIM;}

^specialc {state = SPECIALCHAR;}

[a-zA-Z]+|[0-9]\*|[-,+,%,/,\*,=]|[,;#[]""]|[{,},(,)] {

if(state != UNKNOWN) {

add\_word(yytext, state);

}

else {

switch(find\_word\_type(yytext)) {

case KEYWORD : printf("%s\t -> KEYWORD\n", yytext); break;

case VAR : printf("%s\t -> VARIABLE/IDENTIFIER\n", yytext); break;

case CONST : printf("%s\t -> CONSTANT\n", yytext); break;

case OPERATOR : printf("%s\t -> OPERATOR\n", yytext); break;

case DELIM : printf("%s\t -> DELIMITER\n", yytext); break;

case SPECIALCHAR : printf("%s\t -> SPECIAL CHARACTER\n", yytext); break;

default : printf("%s\t -> NOT IN DICTIONARY\n", yytext);

}

}

}

%%

void main() {

yylex();

}

struct word {

char\* name;

int type;

struct word \* next;

};

struct word \*word\_list;

extern void \*malloc();

void add\_word(char\* word, int state) {

if(word\_list && find\_word\_type(word) != UNKNOWN) {

printf("WARNING : The word '%s' is already defined.\n", word);

return;

}

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

wp->name = (char \* ) malloc(strlen(word) + 1);

strcpy(wp->name, word);

wp->type = state;

wp->next = word\_list;

word\_list = wp;

//print\_list();

}

int find\_word\_type(char\* word) {

struct word \*wp = word\_list;

for(; wp; wp=wp->next) {

if(strcmp(wp->name, word) == 0)

return wp->type;

}

return UNKNOWN;

}

void print\_list() {

struct word \*tmp = word\_list;

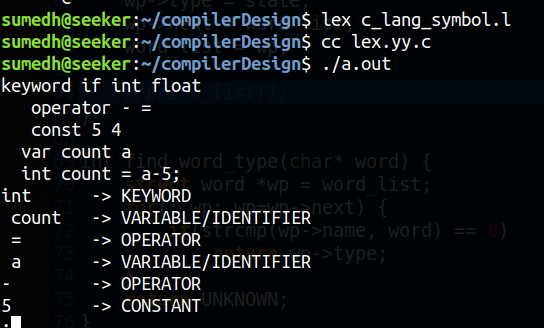
for(;tmp; tmp=tmp->next)

printf("(%s -> %d) -> ", tmp->name, tmp->type);

printf("NULL\n");

}

**Output: 3d**



**Conclusion :**

The lexical tokenizer with and without using symbol table was made to tokenize a subset of english language, and c programming language.