Name: Dhroov Makwana
Gr no.: 22010538
Roll no.: 321042
Batch: A2

Assignment 3

Aim: Write a program to implement a lexical analyser for parts of speech using LEX for subset of English language and C programming language.

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) Write lexical analyser without using Symbol Table for subset of 'C' programming language.

3 d) Write lexical analyser with Symbol Table for subset of 'C' programming language.

Theory:

What is LEX?

- Lex is a program that generates lexical analyzer.
- The lexical analyzer is a program that transforms an input stream into a sequence of tokens.
- It reads the input stream and produces the source code as output through implementing the lexical analyzer in the C program.

LEX Specification

A lex program is divided into 3 sections which are separated by %% delimiters.

```
{ definitions }
%%
 { rules }
patterns {actions}
%%
{ user subroutines }
```

The three sections of a lex program are as follows:

1. **Definition section** : It includes declarations of constant, variable and regular definitions.

2. **Rules section** : This section consists of patterns and actions. The patterns are specified in terms of regular expressions. The actions part describe what action the lexical analyzer should take when it matches a particular pattern. When there are multiple actions to be performed we specify them inside curly braces.

3. **User subroutines section** : It defines a function that is used in one of the actions. It includes the main program that calls required functions.

Functions used in LEX

1. yylex( ) : yylex() is a function of return type int. LEX automatically defines yylex() in lex.yy.c but does not call it. The programmer must call yylex() in the user subroutines section of the LEX program. LEX generates code for the definition of yylex() according to the rules specified in the Rules section.

2. yywrap( ) : LEX declares the function yywrap() of return-type int in the file lex.yy.c . If yywrap() returns zero yylex() assumes there is more input and it continues scanning from the location pointed to by yyin. If yywrap() returns a non-zero value, yylex() terminates the scanning process and returns 0.

Variables used in LEX

Name	Function
yytext	stores recently matched pattern
yyleng	stores length of recently matched pattern
yylval	stores value associated with a token
yymore	append next string matched to current contents of yytext
yyless	remove from yytext all but the first n characters

Compilation steps of LEX program

- **lex filename.l**
- **gcc lex.yy.c**
- **./a.out**

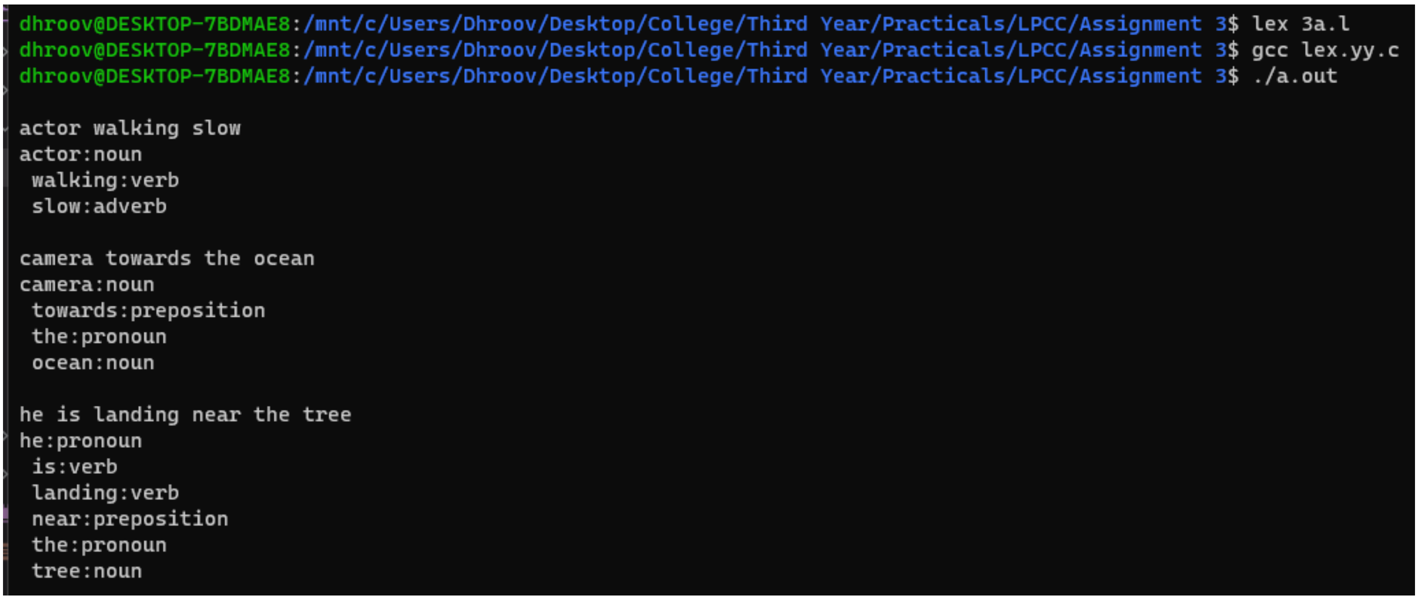
- Firstly lexical analyzer creates a program “lex.1” in the Lex language using command lex file_name.l
- Then Lex compiler runs the lex.1 program and produces a C program “lex.yy.c”.
- Finally, C compiler runs the lex.yy.c program using gcc command and produces an object program “a.out”.
- “a.out” is lexical analyzer that transforms an input stream into a sequence of tokens.

Code 3 A)

```
%{
/* Program to identify parts of speech without Symbol Table */
%}

%%

[\t ]+;

actor |
boy |
camera |
tree |
ocean {printf("%s:noun\n",yytext);}

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

is |
joined |
accepted |
walking |
running |
landing |
packing |
guessing |
jumping |
singing |
performing {printf("%s:verb\n",yytext);}
```

```
soft |
slow |
wealthy |
young |
talented |
rare |
precious |
pretty {printf("%s:adverb\n",yytext);}

about |
at |
on |
since |
to |
into |
over |
upon |
behind |
between |
around |
against |
near |
towards {printf("%s:preposition\n",yytext);}


for |
and |
nor |
but |
or |
yet |
so {printf("%s:conjunction\n",yytext);}
%%

int yywrap(){
return 1;
}

int main(){
printf("\n");
yylex();
```

Output 3 A)



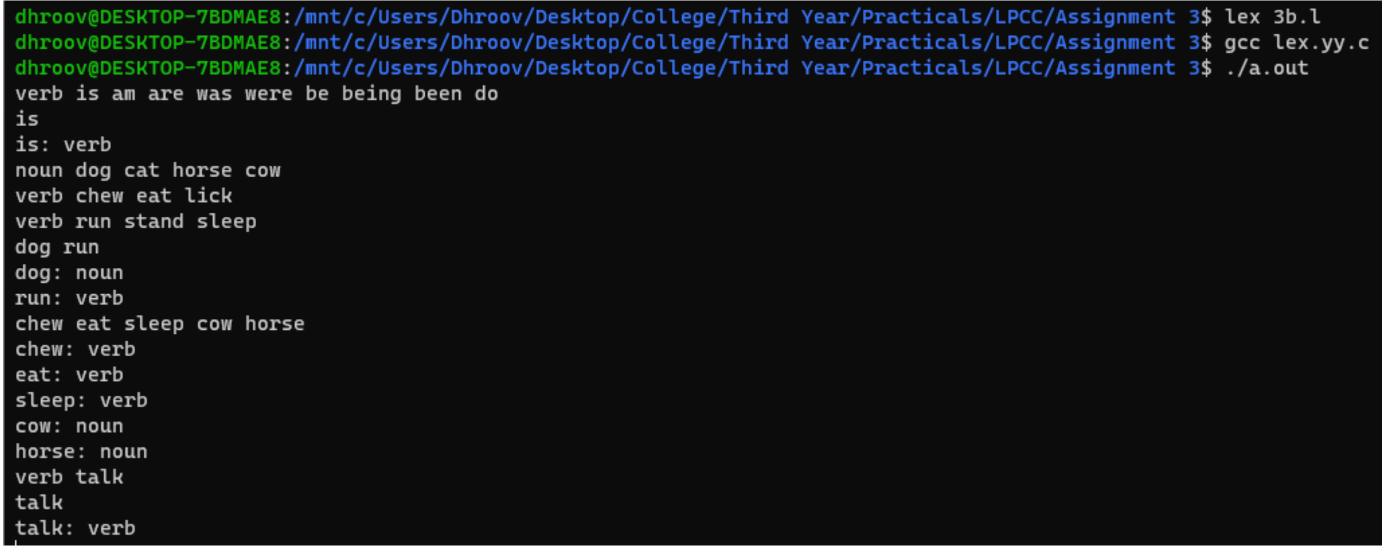
```
dhroov@DESKTOP-7BDMAE8:/mnt/c/Users/Dhroov/Desktop/College/Third Year/Practicals/LPCC/Assignment 3$ lex 3a.l
dhroov@DESKTOP-7BDMAE8:/mnt/c/Users/Dhroov/Desktop/College/Third Year/Practicals/LPCC/Assignment 3$ gcc lex.yy.c
dhroov@DESKTOP-7BDMAE8:/mnt/c/Users/Dhroov/Desktop/College/Third Year/Practicals/LPCC/Assignment 3$ ./a.out

actor walking slow
actor:noun
 walking:verb
 slow:adverb

camera towards the ocean
camera:noun
 towards:preposition
 the:pronoun
 ocean:noun

he is landing near the tree
he:pronoun
 is:verb
 landing:verb
 near:preposition
 the:pronoun
 tree:noun
```

Code 3 B):

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

      /* whenever a line starts with a reserved part of speech name
*/
      /* start defining words of that type */
^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 */ ;

%%

/* 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 */
}

int yywrap(){ return 1;}

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

Output 3 B)

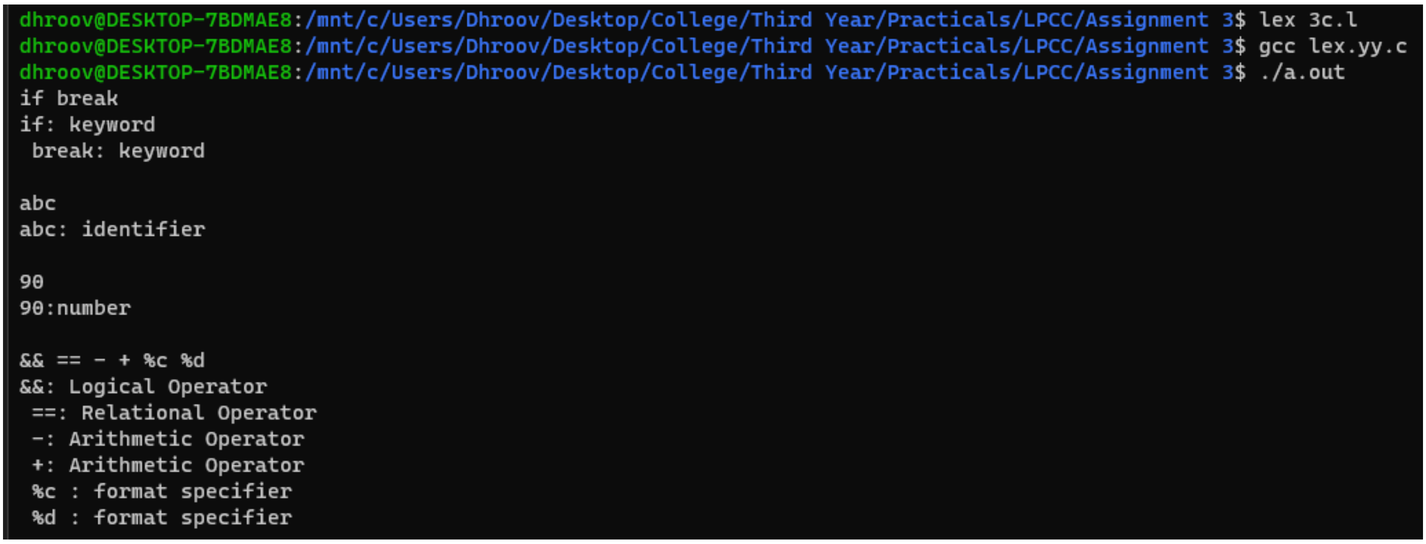


```
dhroov@DESKTOP-7BDMAE8:/mnt/c/Users/Dhroov/Desktop/College/Third Year/Practicals/LPCC/Assignment 3$ lex 3b.l
dhroov@DESKTOP-7BDMAE8:/mnt/c/Users/Dhroov/Desktop/College/Third Year/Practicals/LPCC/Assignment 3$ gcc lex.yy.c
dhroov@DESKTOP-7BDMAE8:/mnt/c/Users/Dhroov/Desktop/College/Third Year/Practicals/LPCC/Assignment 3$ ./a.out
verb is am are was were be being been do
is
is: verb
noun dog cat horse cow
verb chew eat lick
verb run stand sleep
dog run
dog: noun
run: verb
chew eat sleep cow horse
chew: verb
eat: verb
sleep: verb
cow: noun
horse: noun
verb talk
talk
talk: verb
```

Code 3 C):

```
%{
/* Program to identify subset of C programming language without
symbol table */
%}

%%

[\t ]+;

"+" |
"-" |
"*" |
"/" |
"%" {printf("%s: Arithmetic Operator\n",yytext);}

break |
continue |
for |
if |
else |
do |
while {printf("%s: keyword\n",yytext);}
^[a-zA-Z_][a-zA-Z0-9_]+ {printf("%s: identifier\n",yytext);}

"<" |
">" |
">=" |
"<=" |
"==" {printf("%s: Relational Operator\n",yytext);}
"&&" |
"||" {printf("%s: Logical Operator\n",yytext);}
[0-9]+ {printf("%s:number\n",yytext);}
"%d" |
"%s" |
"%f" |
"%c" {printf("%s : format specifier\n",yytext);}

%%

int yywrap()
{
return 1;
}

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

Output 3 C

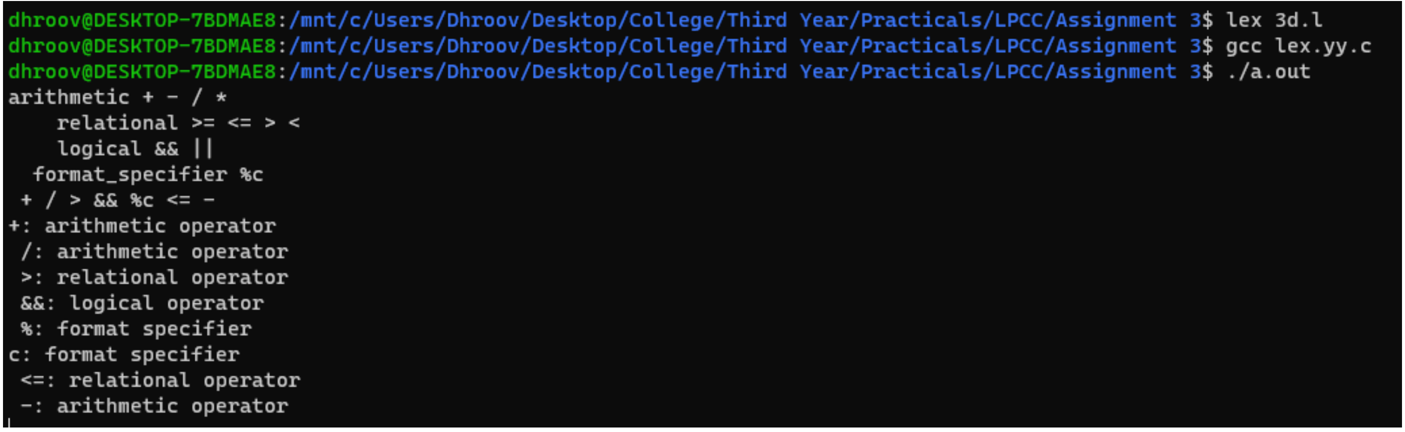


```
dhroov@DESKTOP-7BDMAE8:/mnt/c/Users/Dhroov/Desktop/College/Third Year/Practicals/LPCC/Assignment 3$ lex 3c.l
dhroov@DESKTOP-7BDMAE8:/mnt/c/Users/Dhroov/Desktop/College/Third Year/Practicals/LPCC/Assignment 3$ gcc lex.yy.c
dhroov@DESKTOP-7BDMAE8:/mnt/c/Users/Dhroov/Desktop/College/Third Year/Practicals/LPCC/Assignment 3$ ./a.out
if break
if: keyword
 break: keyword

abc
abc: identifier

90
90:number

&& == - + %c %d
&&: Logical Operator
 ==: Relational Operator
 -: Arithmetic Operator
 +: Arithmetic Operator
 %c : format specifier
 %d : format specifier
```

Code 3 D):

```
%{
/* Program to identify subset of C programming language using
symbol table */

enum{
lookup=0, //searching instead of defining REL_OPTR,
KEYWORD,
ARITH_OPTR,
FORMAT_SPEC,
NUM,
LOGIC_OPTR,
REL_OPTR
};

int state;

int add_subset(int type, char *word);
int search_subset(char *word);

%}

%%

\n {state = lookup;}

^arithmetic {state = ARITH_OPTR;}
^logical    {state = LOGIC_OPTR;}
^relational {state = REL_OPTR;}
^number     {state = NUM;}
^keyword    {state = KEYWORD;}
^format_specifier {state = FORMAT_SPEC;}


[a-zA-Z]+ |
[0-9]+ |
[+*/%-] |
"<=" |
">=" |
"<" |
">" |
"&&" |
“%c” |
“%d” |
“%f” |
"||" {
if(state != lookup)
{
    add_subset(state, yytext);
}
else
{
    switch(search_subset(yytext))
    {
        case ARITH_OPTR:
            printf("%s: arithmetic operator\n",yytext);
            break;
        case REL_OPTR:
            printf("%s: relational operator\n",yytext);
            break;
        case LOGIC_OPTR:
            printf("%s: logical operator\n",yytext);
            break;
        case NUM:
            printf("%s: number\n",yytext);
            break;
        case KEYWORD:
            printf("%s: keyword\n",yytext);
            break;
        case FORMAT_SPEC:
            printf("%s: format specifier\n",yytext);
            break;
        default: printf("%s: not a subset\n",yytext);
    }
}
}

%%

struct word{
    char *word_name;
    int word_type;
    struct word *next;
};

struct word *first;
extern void *malloc();
int add_subset(int type, char *word)
{
    struct word *wp;
    if(search_subset(word) != lookup)
    {
        printf("%s is already defined\n",word);
        return 0;
    }

    wp = (struct word *) malloc(sizeof(struct word));
    wp->next = first;

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

int search_subset(char *word)
{
    struct word *wp= first;
    while(wp){
        if(strcmp(wp->word_name, word)==0) return wp->word_type;
        wp=wp->next;
    }

    return lookup;
}

int yywrap(){ return 1;}

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

Output 3 D)



```
dhroov@DESKTOP-7BDMAE8:/mnt/c/Users/Dhroov/Desktop/College/Third Year/Practicals/LPCC/Assignment 3$ lex 3d.l
dhroov@DESKTOP-7BDMAE8:/mnt/c/Users/Dhroov/Desktop/College/Third Year/Practicals/LPCC/Assignment 3$ gcc lex.yy.c
dhroov@DESKTOP-7BDMAE8:/mnt/c/Users/Dhroov/Desktop/College/Third Year/Practicals/LPCC/Assignment 3$ ./a.out
arithmetic + - / *
    relational >= <= > <
    logical && ||
  format_specifier %c
 + / > && %c <= -
+: arithmetic operator
 /: arithmetic operator
 >: relational operator
 &&: logical operator
 %: format specifier
c: format specifier
 <=: relational operator
 -: arithmetic operator
```

Conclusion: I was successful in implementing lex programs for identifying parts of speech of English language and subset of C programming language, with and without symbol table.