VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on COMPILER DESIGN

Submitted by

MAHAVIR NAHATA(1BM21CS100)

Under the Guidance of

Prof. Prameetha Pai Assistant Professor, BMSCE

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

November 2023-February 2024

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**Compiler Design**" carried out by Mahavir Nahata**(1BM21CS100)** , who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023-24.

The Lab report has been approved as it satisfies the academic requirements in respect of **Compiler Design-(22CS5PCCPD)** work prescribed for the said degree.

Prof. Prameetha Pai	Dr. Jyothi Nayak
Assistant professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

B. M. S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

I, Mahavir Nahata (1BM21CS100), student of 5th Semester, B.E, Department of Computer Science and Engineering, B. M. S. College of Engineering, Bangalore, here by declare that, this lab report entitled " **Compiler Design**" has been carried out by me under the guidance of Prof. Prameetha Pai, Assistant Professor, Department of CSE, B. M. S. College of Engineering, Bangalore during the academic semester November-2023-February-2024.

I also declare that to the best of my knowledge and belief, the development reported here is not from part of any other report by any other students.

TABLE OF CONTENTS

9	
9.1	Write a program in YACC to generate three address code for a given expression.

04/11/23

(A) lex (programname.l) → in cmt

gcc lex@.yy.c

~~a.out~~ ./a.out

(*) yywrap → i/p stream is over

Q1) Write a lex program to identify datatype - int, char, float & variables:

```
%{
#include<stdio.h>
%}
%%
int|char|float { printf("%s is keyword", yytext); }
[a-zA-Z] [_ 0-9 a-z A-Z]* { printf("%s is variable", yytext); }
%%
int yywrap(){
    return 0; }

int main(){
  yylex();
  return 0;
}
```

[0-9] [space] [_ 0-9 a-z A-Z]*

{ printf("%s is invalid token", yytext); }

output:

- int
  int is keyword
- char
  char is keyword
- float
  float is keyword
- 1 int
  1 is invalid token int is keyword.
  1 is invalid token

(2) Write a lex program to identify each character as a consonant or vowel in a given sentence :

```
%{
#include<stdio.h>
%}
%%
[aeiouAEIOU] {printf("%s is vowel\n", yytext);}
[a-zA-Z] {printf("%s is consonant\n", yytext);}
%%
int yywrap() {
    return 0; }
int main()
{
 yylex();
 return 0;
}
```

Output :

```
Compiler Design
No. of vowels and consonants are 5 and 9.
This is a book
No. of vowels and consonants are 5 and 6
```

3) Write lex program to identify alphabets as characters & numbers or digits.

Expected output:

Enter input : sum 33
sum is stream of characters
33 is digit

Solⁿ

```
%{
#include<stdio.h>
%}
%%
[a-zA-Z]*   printf("%s is character", yytext);
[0-9]*      printf("%s is digit", yytext);
%%
int yywrap(){
    return 0;}

int main()
{
  yylex();
  return 0;
}
```

3 completed

Output:

```
33
33 is digit
sum
sum is alphabet
sum 33
sum is character 33 is digit
```

Sun
18/11/23.

4) Write a lex program to count no. of words in an i/p sentence.

```
%{
#include<stdio.h>
%}
%%
[a-zA-Z0-9]*    {c=c+1;}
"\n"            {printf("%d\n",c); c=0;}
%%
int yywrap()
{
return 0;
}
int main()
{
  yylex();
  return 0;
}
```

output :

```
hello m
2
mahavir goahata is a good boy
6
```

Lab program:

a) Write a program in LEX to recognize floating point number:

```
%{
#include <stdio.h>
#include <string.h>
%}
%%
[+-]?[0-9]+[.][0-9]+ {printf("floating point no.");}
[+-]?[0-9]+ {printf("integer");}
%%
int yywrap(){
    return 0;
}

int main()
{
  yylex();
  return 0;
}
```

Sun 25/11/23

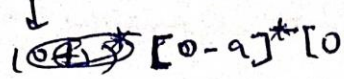
2 programs solved

Output:

```
+22.7
floating point no.

-23.7
floating point no.

22
integer
```

① Write a Lex program to recognize the following tokens over the alphabets {0,1,...,9}:

(a) set of all strings ending in 00.



```
%{
#include <stdio.h>
#include <string.h>
%}
%%
[0-9]* [0][0]   { printf("valid"); }
%%
int yywrap()
{
  return 0;
}
int main()
{
   yylex();
   return 0;
}
```

• output:

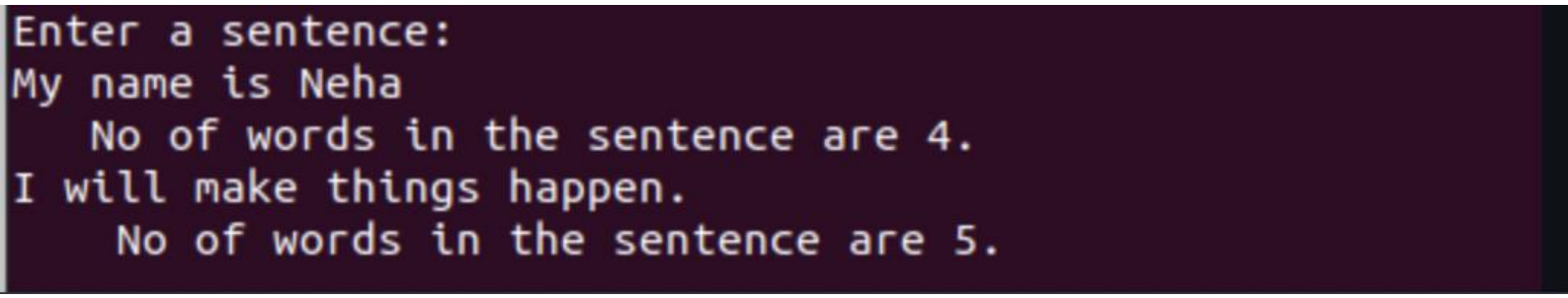
```
10 0                    001
valid                   valid 1

0011 00                 0011
valid                   valid 11
```

26/12/23

1.1 **Write a program in lex to count the number of words in a sentence. Code**

```
%{
#include<stdio.h>

 int words;
%}
%%
[^\t\n ]+ {words++;}
\n {printf("No of words in the sentence are %d.\n",words),words=0;}
%%
void main()
{
printf("Enter a sentence:\n"); yylex();
}
int yywrap()
{
return 1;
}
```

Output:



1.3 **Write a program in lex to identify tokens in a program by taking input from a file and printing the output on the terminal.**

Cod::

```
%{
#include<stdio.h>
%}
%%
Char |int |float {printf("%s is a keyword.\n",yytext);}
[a-zA-Z][a-zA-Z0-9]* {printf("%s is an identifier.\n",yytext);}
, {printf("%s is a separator.\n",yytext);}
; {printf("%s is a delimiter.\n",yytext);}
"=" {printf("%s is an assignment operator.\n",yytext);}

"+"|"-"|"*"|"/" {printf("%s is a binary operator.\n",yytext);}
[0-9]+ {printf("%s is/are digit(s).\n",yytext);}
\n ;
%%
void main()
{
yyin=fopen("input.txt","r");

yylex();
fclose(yyin);
}
int yywrap()
{
return 1;
}
```

Output:

```
int is a keyword.
 sum is an identifier.
, is a separator.
x is an identifier.
= is an assignment operator.
2 is/are digit(s).
, is a separator.
y is an identifier.
= is an assignment operator.
3 is/are digit(s).
; is a delimiter.
sum is an identifier.
= is an assignment operator.
x is an identifier.
+ is a binary operator.
y is an identifier.
; is a delimiter.
```

1.4 **Write a program to read an input sentence and to check if the sentence begins with English articles (A, a,AN,An,THE and The):**

Code::

```
%{
#include<stdio.h> int flag=0;
%}
%%
^(an|An|The|the|A|a)[" "].* {flag=1;}
.* {flag=0;}
\n {return 0;}
%%
int yywrap()
{
return 1;
}
void main()
{
printf("Enter a sentence:\n"); yylex();
if(flag==1)
printf("Starts with an article!\n"); else
printf("Does not start with an article!\n");
}
```

Output::

```
Enter a sentence:
A book is lying on the table.
Starts with an article!
```

```
Enter a sentence:
An apple a day keeps the doctor away.
Starts with an article!
Enter a sentence:
The sun rises in the east.
Starts with an article!
```

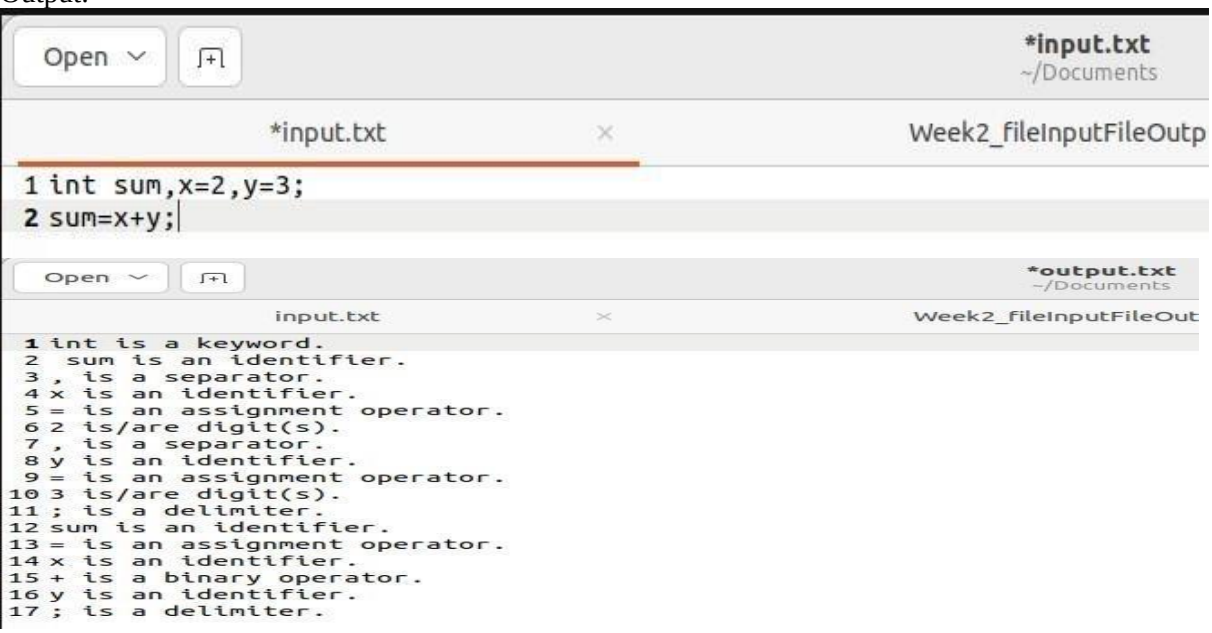
1.4 **Write a program in lex to identify tokens in a program by taking input from a file and printing the output in another file.**

Code

```
%{
#include<stdio.h>
%}
%%
char|int|float {fprintf(yyout,"%s is a keyword.\n",yytext);}
[a-zA-Z][a-zA-Z0-9]* {fprintf(yyout,"%s is an identifier.\n",yytext);}
, {fprintf(yyout,"%s is a separator.\n",yytext);}
; {fprintf(yyout,"%s is a delimiter.\n",yytext);}
"=" {fprintf(yyout,"%s is an assignment operator.\n",yytext);}

"+"|"-"|"*"|"/" {fprintf(yyout,"%s is a binary operator.\n",yytext);}

 [0-9]+ {fprintf(yyout,"%s is/are digit(s).\n",yytext);}
\n ;
%%
void main()
{
yyin=fopen("input.txt","r");

yyout=fopen("output.txt","w");

yylex();
printf("Printed in output.txt\n");
```

```
fclose(yyin);
fclose(yyout);
}
int yywrap()
{
return 1;
}
```

Output:



Q)Write a program in lex to find the length of the input string.

Code:

```
%{
#include<stdio.h>
%}
%%
[a-zA-Z0-9.,!? \t]+ {printf("Length of input string is %d.\n",yyleng);}
%%
void main()
{
printf("Enter a string:\n");

yylex();
}
int yywrap()
{
return 1;
}
```

Output:

```
neha29@neha-VirtualBox:~/Documents$ lex Week2_lengthofString.l
neha29@neha-VirtualBox:~/Documents$ gcc lex.yy.c
neha29@neha-VirtualBox:~/Documents$ ./a.out
Enter a string:
Good Morning!
Length of input string is 13.

Where do you stay?
Length of input string is 18.
```

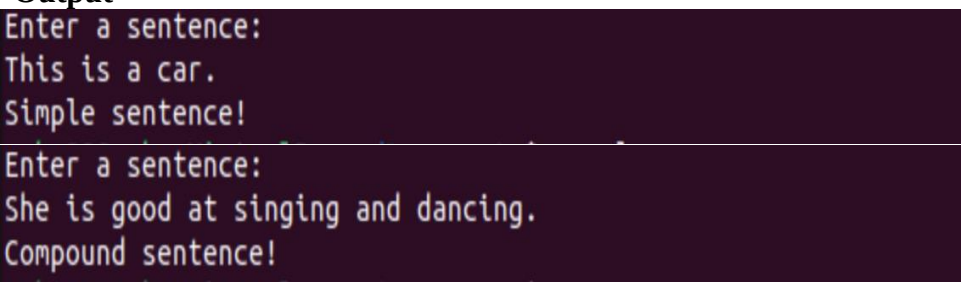
1.2 **Read and input sentence, and check if it is compound or simple. If a sentence has the word- and , or ,but ,because ,if ,then ,nevertheless then it is compound else it is simple.**

Code::

```
%{
#include<stdio.h>

int flag=0;
%}
%%
If |then |but |because |nevertheless |and |or {flag=1;}
. ;
\n {return 0;}
%%
int yywrap()
{
return 1;
}
void main()
```

```
{
printf("Enter a sentence:\n");

yylex();
if(flag==1)
printf("Compound sentence!\n");

else
printf("Simple sentence!\n");
}
```

Output



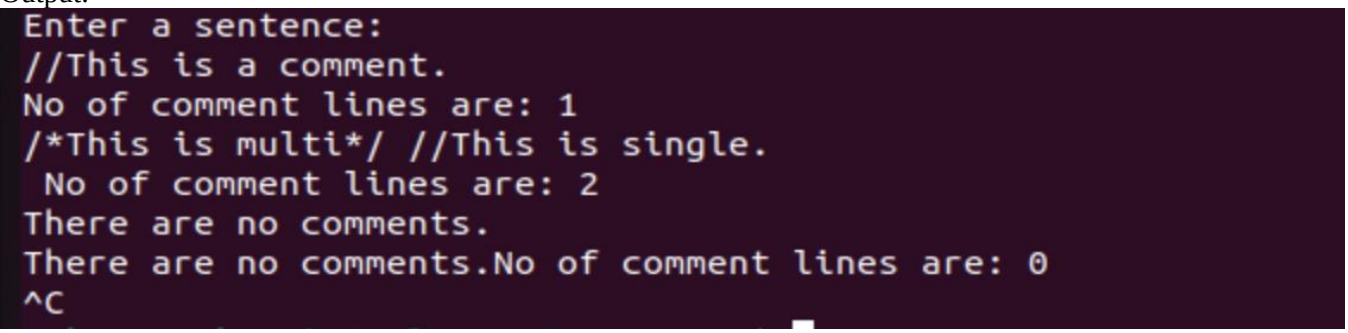
1.5 **Lex program to count the number of comment lines (multi line comments or single line) in a program. Read the input from a file called input.txt and print the count in a file called output.txt.**

Code

```
%{
#include<stdio.h>

 int c=0;
%}
%%
"\/\*"[^*]*\*+([^/*][^*]*\*+)*\/ {c++;}

 "//".* {c++;}
. ECHO;
%%
int yywrap()
{
return 1;
```

```
}
void main()
{
yyin=fopen("input.txt","r");

yyout=fopen("output.txt","w");

yylex();
printf("The number of comments are:%d\n",c);
fclose(yyin);
fclose(yyout);
}
```

Output:



(b) Set of all strings with three consecutive 222's.

Logic:
[0-9]*[2][2][2][0-9]*.

Sol:

```
%{
#include<stdio.h>
#include<string.h>
%}
%%
[0-9]*[2][2][2][0-9]*    {printf("valid");}
%%
int yywrap()
{
return 0;
}
int main()
{
yylex();
return 0;
}
```

output:

222
→ valid

1222
→ valid

36222945
→ valid

222 1 222
→ valid

2131
→ 2131

(d) Set of all strings such that 10th symbol from the right end is 1. (binary)

⇒ $(0+1)^* 1 (0+1)^9$

```
%%
[0-9]*1[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]
                    { printf("valid"); }
%%
int yywrap()
{
 return 0;
}
int main()
{
  yylex();
 return 0;
}
```

output:

1010686917 45

→ valid

1076 64867

→ 0

2134567 8922

→ valid

(c) Set of all strings such that every block of consecutive symbols contain atleast two 5's.

sol°

```
%{
#include<stdio.h>
#include<math.h>
int value=0, i=0, j=0, flag=0;
%}
%%
1[01]  { for(i=yyleng-1; i>0; i--)
         {
          val = val + (yytext[i]-8) * pow(2,j)
          j++;
         }
       }
if(value%5==0)
    flag=1;
}
[\n] return 0;
%%
int main()
{
  yylex();
   if(flag==1)
    {
     success!
    }
}
```

8) set of all 4 digits nombes whose sum is 9 :

Sol

% %

[0-9][0-9][0-9][0-9]

```
int value=0;
{ for (i=0; i<4; i++)
  {
    value = value + (yytext[i]-48);
  }
if (value==9)
  {
    printf("Number %s
    matches the
    condition", yytext);
  }
}
```

output :

1 2 3 3

sum = 9

1 4 3 1

sum = 9

8) Write a program to perform Recursive descent parsing on the following grammar: $S \rightarrow cAd$, $A \rightarrow ab \mid a$.

Sol:

```
main()
{
  S();
  if (lookahead == '$')
  {
    printf("success");
  }
}
S()
{
  if(lookahead == 'c')
  {
    match('c');
    A();
    match('d');
  }
}
A()
{
  if (lookahead == 'ab')
  {
    match('a'); match('b');
  }
  else if (lookahead == 'a')
  {
    match('a');
  }
  else
    return();
}
```

```
match(char c)
{
  if(lookahead == c)
    lookahead = getchar();
  else
    printf("error");
}
```

cad$

cad$

caid$

cabd ✓

cad ✓

d Invalid

```
char lookahead;

void match(char c)
{
  if (lookahead == c)
  {
    lookahead = getchar();
  }
  else
  {
    printf("Error");
    exit(1);
  }
}

void A()
{
  if (lookahead == 'a')
  {
    match('a');
    if (lookahead == 'b')
    {
      match('b');
    }
  }
  else{
    printf("Error\n");
    exit(1);
  }
}

void S()
{
  if (lookahead == 'c')
  {
    match('c');
    A();
    match('d');
  }
  else{
```

Sun 26/12/23

```
int main()
{
  lookahead = getchar();
  S();
  if (lookahead == '$')
  {
    printf("Success\n");
  }
  else {
    printf("Error");
  }
  return 0;
}
```

output:

```
Enter string : cad
string is accepted

Enter string : cabd
String is accepted
```

soln

```
%{
#include "y.tab.h"
external int yytext;
%}

%%
[0-9]+ { yyval=atoi (yytext); return digit;}
[\t];
[\n] return 0;
return yytext[0];
%%
int yywrap(){
}
```

Prog.2:

$E \rightarrow E+T/E-T/T$

$T \rightarrow T*F/T/F/F$

$F \rightarrow (E)/digit$

```
%{
#include<ctype.h>
#include<stdio.h>
#include<stdlib.h>
%}
%token digit
%%
S: E {printf("\n%d",$1);};
E: E'+'T {$$=$1+$3;}
 | E'-'T {$$=$1-$3;}
 | T
 ;
T: T'*'F {$$=$1*$3;}
 | T'/'F {$$=$1/$3;}
 | F
 ;
F: F'^'P {$$=pow($1,$3);}
 | P
```

```
%% E %
    | digit   { printf("%d", $1); }
    ;
%%
int main()
{
   printf("Enter infix expr : ");
   yy
   return 0;
}

void yyerror()
{
   printf("Error \n");
}
```

e.g. 2+5*3
= 253*+

output:

Enter infix expression :
4 + 3 * (5 - 9) / 4 ^ 4
=> 4359 -

Sun 23/1/24

Q) Write a C program to design lexical analysis to recognize any five keywords, identifiers, numbers, operators and punctuations:

```
#include <stdio.h>

 #include <string.h>

 #include <ctype.h>


void lexicalAnalyzer(char input_code[]) {
   char *keywords[] = {"if", "else", "while", "for", "return"};
   char *operators[] = {"+", "-", "*", "/", "=", "==", "<", ">", "<=", ">="};
   char *punctuations[] = {",", ";", "(", ")", "{", "}"};


   char *token = strtok(input_code, " \t\n");


   while (token != NULL)
    {
   if (isdigit(token[0]))
    {
        printf("Number: %s\n", token);
     }

     else if (isalpha(token[0]) || token[0] == '_')

     {

      int isKeyword = 0;
        for (int i = 0; i < sizeof(keywords) / sizeof(keywords[0]); i++)
        {
```

```
        if (strcmp(token, keywords[i]) == 0) {
            printf("Keyword: %s\n", token); isKeyword = 1;
            break;
          }
        }
        if (!isKeyword) { printf("Identifier: %s\n", token);
        }
      }

      else if (strchr("+-*/=<>(){}[]", token[0]) != NULL) { printf("Operator: %s\n", token);
      }
      else if(strchr(",;", token[0]) != NULL)
      {
        printf("Punctuation:%s\n",token);
      }
      token = strtok(NULL, " \t\n");
    }
}
int main() {
    char input_code[] = "if ( x > 0 ) { return x ; } else { return -x ; }";

    lexicalAnalyzer(input_code);
    return 0;
}
```

Output::

ain.c

Save Run Output

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void lexicalAnalyzer(char input_code[]) {
    char *keywords[] = {"if", "else", "while", "for", "return"};
    char *operators[] = {"+", "-", "*", "/", "=", "==", "<", ">", "<=", ">="};
    char *punctuations[] = {",", ";", "(", ")", "{", "}"};

    char *token = strtok(input_code, " \t\n");

    while (token != NULL) {
        if (isdigit(token[0])) {
            printf("Number: %s\n", token);
        } else if (isalpha(token[0]) || token[0] == '_') {
            int isKeyword = 0;
            for (int i = 0; i < sizeof(keywords) / sizeof(keywords[0]); i++) {
                if (strcmp(token, keywords[i]) == 0) {
                    printf("Keyword: %s\n", token);
                    isKeyword = 1;
                    break;
                }
            }
            if (!isKeyword) {
                printf("Identifier: %s\n", token);
            }
        } else if (strchr("+-*/=<>(){}[]", token[0]) != NULL) {
            printf("Operator: %s\n", token);
        }
        else if(strchr(",;", token[0]) != NULL)
        {
            printf("Punctuation:%s\n",token);
```

```
/tmp/AWz7FTr3It.o
Keyword: if
Operator: (
Identifier: x
Operator: >
Number: 0
Operator: )
Operator: {
Keyword: return
Identifier: x
Punctuation:;
Operator: }
Keyword: else
Operator: {
Keyword: return
Operator: -x
Punctuation:;
Operator: }
```

Lab 7

1.1 **Write a program in YACC to design a suitable grammar for evaluation of arithmetic expression having +, -, * and /.**

Code

LEX

```
%{
#include<stdio.h>

 #include<stdlib.h>

 #include "y.tab.h"

extern int yylval;
%}
%%
[0-9]+ {yylval=atoi(yytext);return num;}

[\t ] ;
```

```
\n {return 0;}
. {return yytext[0];}
%%
int yywrap()
{
}
```

YACC

```
%{
#include<stdio.h>
 #include<stdlib.h>
int yyerror(const char *s);
int yylex(void);
%}
%token num;
%left '+' '-'
%left '*' '/'
%left ')'
%left '('
%%
s:e {printf("Valid expression!\n");
printf("Result:%d\n",$$); exit(0);
   }
;
e:e'+'e {$$=$1+$3;}
|e'-'e {$$=$1-$3;}
|e'*'e {$$=$1*$3;}
|e'/'e {$$=$1/$3;}
|'('e')' {$$=$2;}
|num {$$=$1;}
;
%%
```

```
void main()
{
printf("Enter an arithmetic expression:\n"); yyparse();
}
int yyerror(const char *s)
{
printf("Invalid expression!\n"); return 0;
}
```

Output:

1.2 **Write a program in YACC to recognize strings of the form {(a^n)b , n>=5}. Code**

LEX

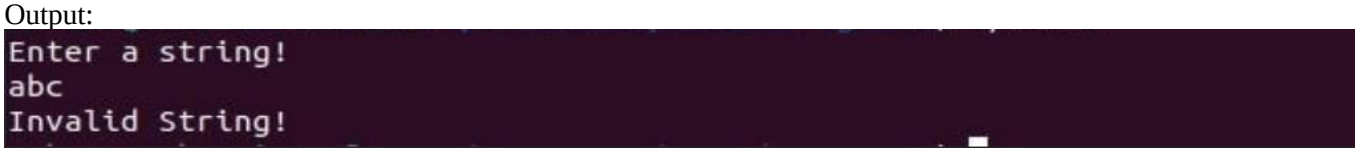
```
%{
#include<stdio.h>

#include<stdlib.h>

#include "y.tab.h"

extern int yylval;
%}
%%
[aA] {yylval=yytext[0];return A;}

[bB] {yylval=yytext[0];return B;}
```

```
\n {return NL;}
. {return yytext[0];}
%%
int yywrap()
{
return 1;
}
```

YACC

```
%{
#include<stdio.h>
 #include<stdlib.h>
int yyerror(char *s);
int yylex(void);
%}
%token A
%token B
%token NL
%%
smtr:A A A A A S B NL {printf("Parsed using the rule (a^n)b, n>=5.\nValid String!\n");}
;
%%
void main()
{
printf("Enter a string!\n"); yyparse();
}
int yyerror(char *s)
{
printf("Invalid String!\n"); return 0;
}
```

Output:



```
Enter a string!
aaaaaaab
Parsed using the rule (a^n)b, n>=5.
Valid String!
ab
Invalid String!
```

1.3 **Write a program in YACC to generate syntax tree for a given arithmetic expression. Code**

LEX

```
%{
#include<stdio.h>

#include<stdlib.h>

#include "y.tab.h"

extern int yylval;
```

```
%}
%%
[0-9]+ {yylval=atoi(yytext);return digit;}

[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
return 1;
}
```

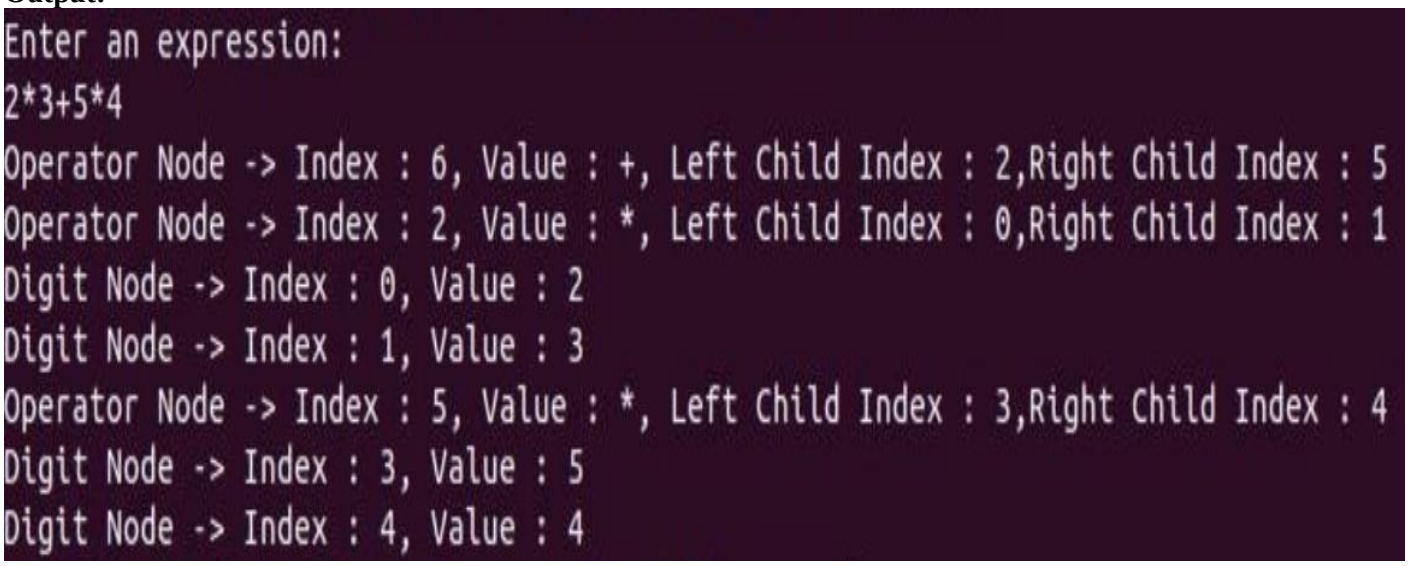
YACC

```
%{
#include <math.h>

#include<ctype.h>

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int yyerror(char *s);

 int yylex(void);

struct tree_node
{
char val[10];

int lc;
int rc;
};
int ind;
struct tree_node syn_tree[100];

void my_print_tree(int cur_ind);
```

```
int mknode(int lc,int rc,char *val);
%}
%token digit
%%
S:E {my_print_tree($1);}
;
E:E'+'T {$$=mknode($1,$3,"+");}
|T {$$=$1;}
;
T:T'*'F {$$= mknode($1,$3,"*");}
|F {$$=$1;}
;
F:'('E')' {$$=$2;}
|digit {char buf[10];sprintf(buf,"%d", yylval);$$ = mknode(-1,-1,buf);}
;
%%
int main()
{
ind=0;
printf("Enter an expression:\n"); yyparse();
return 0;
}
int yyerror(char *s)
{
printf("NITW Error\n"); return 0;
}
int mknode(int lc,int rc,char val[10])
{
strcpy(syn_tree[ind].val,val);
syn_tree[ind].lc = lc;
syn_tree[ind].rc = rc; ind++;
return ind-1;
}
```

```
/*my_print_tree function to print the syntax tree in DLR fashion*/ void my_print_tree(int
cur_ind)
{
if(cur_ind==-1) return;
if(syn_tree[cur_ind].lc==-1&&syn_tree[cur_ind].rc==-1)
printf("Digit Node -> Index : %d, Value : %s\n",cur_ind,syn_tree[cur_ind].val); else
printf("Operator Node -> Index : %d, Value : %s, Left Child Index : %d,Right Child Index :
%d\n",cur_ind,syn_tree[cur_ind].val, syn_tree[cur_ind].lc,syn_tree[cur_ind].rc);
my_print_tree(syn_tree[cur_ind].lc);
my_print_tree(syn_tree[cur_ind].rc);
}
```

Output:



Write a program in YACC to generate three address code for a given expression. Code

LEX

```
%{
#include<stdio.h>

#include<stdlib.h>

#include"y.tab.h"

extern int yylval;

extern char iden[20];
%}
d [0-9]+
a [a-zA-Z]+
%%
{d} { yylval=atoi(yytext); return digit; }
{a} { strcpy(iden,yytext); yylval=1; return id;}

 [ \t]

{;}
\n return 0;
. return yytext[0];
%%
int yywrap()
{
return 1;
}
```

YACC

```
%{
```

```
#include <math.h>

#include<ctype.h>

#include<stdio.h>

int yyerror(char *s);

 int yylex(void);
int var_cnt=0;
char iden[20];
%}
%token id
%token digit
%%
S:id '=' E {printf("%s=t%d\n",iden,var_cnt-1);}
E:E '+' T {$$=var_cnt; var_cnt++; printf("t%d = t%d + t%d;\n", $$, $1, $3 );}
|E '-' T { $$=var_cnt; var_cnt++; printf("t%d = t%d - t%d;\n", $$, $1, $3 );}
|T {$$=$1;}
;
T:T '*' F {$$=var_cnt; var_cnt++; printf("t%d = t%d * t%d;\n", $$, $1, $3 );}
|T '/' F {$$=var_cnt; var_cnt++; printf("t%d = t%d / t%d;\n", $$, $1, $3 );}
|F {$$=$1;}
;
F:P '^' F {$$=var_cnt; var_cnt++; printf("t%d = t%d ^ t%d;\n", $$, $1, $3 );}
|P {$$ = $1;}
;
P: '(' E ')' {$$=$2;}
|digit {$$=var_cnt; var_cnt++; printf("t%d = %d;\n",$$,$1);}
;
%%
int main()
{
var_cnt=0;
printf("Enter an expression:\n"); yyparse();
```

```
return 0;
}
int yyerror(char *s)
{
printf("Invalid expression!"); return 0;
}
```

Output: