PROGRAM-1

1. Write a LEX program to recognize valid arithmetic expression. Identifiers in the expression could be only integers and operators could be + and *. Count the identifiers & operators present and print them separately.

P1.l

```
%{
        #include<stdio.h>
        int num=0, op=0, i=0, j=0, number[10], top=-1;
        char opr[10], stack[10];
%}
%%
[0-9]+ {number[num++]=atoi(yytext);}
"+" { opr[op++]='+';}
"*" { opr[op++]='*';}
"(" { stack[++top]='(';}
")" {
        if(stack[top]=='(' && top!=-1)
                top--;
        else{
                printf("Invalid expression\n");
                exit(0);
        }
}
%%

void main()
{
        printf("Enter Expression:\n");
        yylex();

        if(top==-1 && num==op+1) {
                printf("Valid Expression:\n");
                printf("Number of identifiers=%d\n",num);
                printf("Identifier are present\n");

                for(i=0;i<num;i++) {
                        printf("%d\n",number[i]);
                }
                printf("Number of operators=%d\n",op);
                printf("operators present are\n");
                for(j=0;j<op;j++) {
                        printf("%c\n",opr[j]);
                }
        }
```

```
        else {
                printf("Invalid Expression");
        }
}
```

```
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab# cd p1
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p1# gedit p1.l
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p1# lex p1.l
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p1# cc lex.yy.c -ll
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p1# ./a.out
Enter Expression:
2+6

Valid Expression:
Number of identifiers=2
Identifier are present
2
6
Number of operators=1
operators present are
+
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p1#
```

PROGRAM-2

2. Write YACC program to evaluate arithmetic expression involving operators: +, -, *, and /.

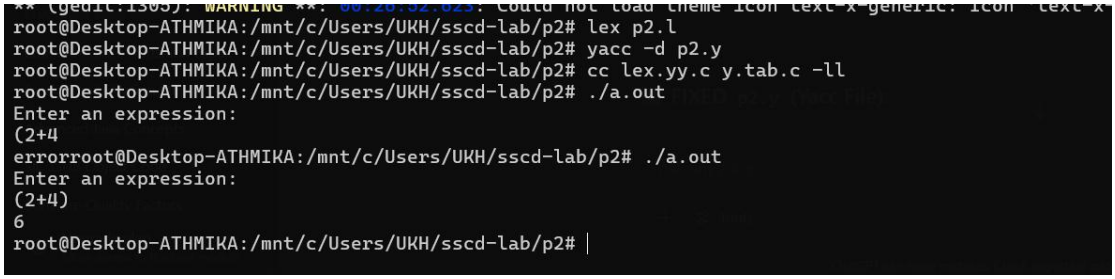
P2.l

```
%{
        #include "y.tab.h"
        extern int yylval;
%}
%%
[0-9]+ {
        yylval=atoi(yytext);
        return num;
    }
[\+\-\*\/] { return yytext[0]; }
[)] { return yytext[0]; }
[(] { return yytext[0]; }
. { ; }
\n { return 0; }
%%
```

P2.y

```
%{
        #include<stdio.h>
        #include<stdlib.h>
        int yylex();
        int yyerror(const char *s);
%}
%token num
%left '+' '-'
%left '*' '/'
%%
input : exp {
                printf("%d\n",$$);
                exit(0);
        }
exp : exp'+'exp { $$=$1+$3; }
        | exp'-'exp{ $$=$1-$3; }
        | exp'*'exp{ $$=$1*$3; }
        | exp'/'exp {
                if($3==0){
                        printf("Divide by Zero error\n");
                    exit(0);
                }
                else
                        $$=$1/$3;
```

```
        }
    | '('exp')' { $$=$2; }
    | num{ $$=$1; };
%%
int yyerror(const char *s)
{
        printf("error");
        exit(0);
}

int main()
{
        printf("Enter an expression:\n");
        yyparse();
        return 0;
}
```



PROGRAM-3

3. Develop, Implement and Execute a program using YACC tool to recognize all strings ending with b preceded by n a's using the grammar an b (note: input n value).

P3.l

```
%{
        #include "y.tab.h"
%}
%%
a  { return A; }
b  { return B; }
.  { return yytext[0]; }
\n { return 0; }
%%
```

P3.y

```
%{
        #include<stdio.h>
        #include<stdlib.h>
        int yylex();
        int yyerror();
%}
%token A B
%%
S : X B
X : A X
  | ;
%%
void main()
{
        printf("Enter the input\n");
        yyparse();
        printf("Valid String\n");
}
int yyerror()
{
        printf("Invalid String\n");
        exit(0);
}
```

```
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p3# lex p3.l
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p3# yacc -d p3.y
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p3# cc lex.yy.c y.tab.c -ll
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p3# .a/.out
-bash: .a/.out: No such file or directory
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p3# ./a.out
Enter the input
aab
Valid String
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p3#
```

PROGRAM-4

4. Design, develop and implement YACC/C program to construct Predictive / LL(1) Parsing Table for the grammar rules: A ->aBa , B-> bB | . Use this table to parse the sentence: abba$.

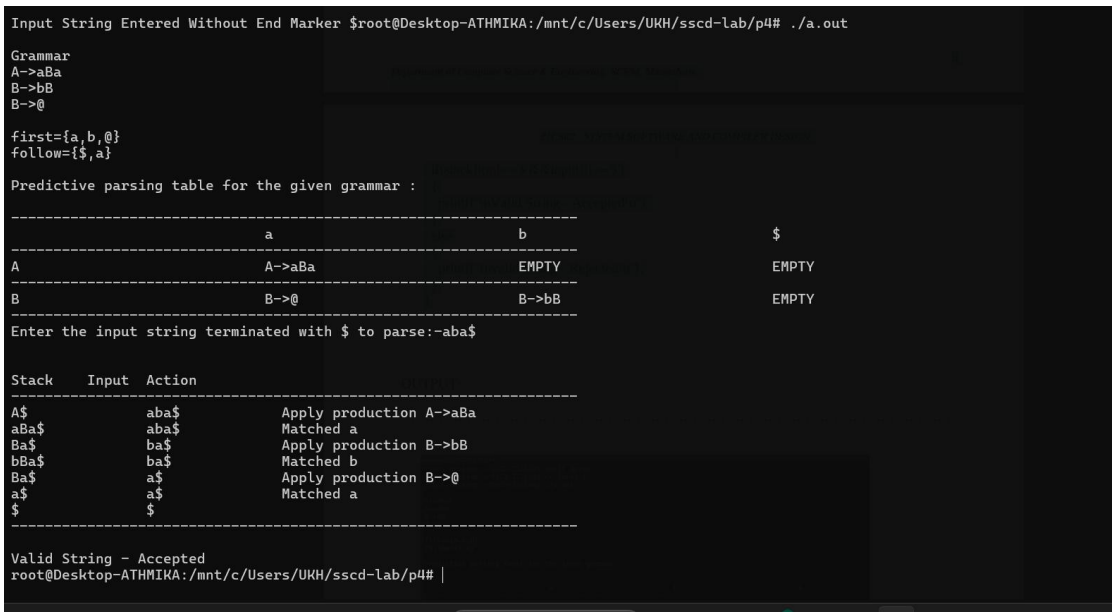
P4.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

char prod [3][10]={"A->aBa","B->bB","B->@"};
char first[3][10]={"a","b","@"};
char follow[3][10]={"$","a","a"};
char table[3][4][10];

char input[10];
int top=-1;
char stack[25];
char curp[20];

void push(char item)
{
  stack[++top]=item;
}
void pop()
{
  top=top-1;
}
void display()
{
  int i;
  for(i=top;i>=0;i--)
  printf("%c",stack[i]);
}

int numr(char c)
{
  switch(c)
  {
    case'A':return 1;
    case'B':return 2;
    case'a':return 1;
    case'b':return 2;
    case'@':return 3;
  }
  return 1;
}
```

```
int main()
{
 char c;
 int i,j,k,n;
 for(i=0;i<3;i++){
  for(j=0;j<4;j++){
   strcpy(table[i][j],"EMPTY");
  }
 }
 printf("\nGrammar\n");

 for(i=0;i<3;i++)
 printf("%s\n",prod[i]);

 printf("\nfirst={%s,%s,%s}",first[0],first[1],first[2]);
 printf("\nfollow={%s,%s}\n",follow[0],follow[1]);
 printf("\nPredictive parsing table for the given grammar :\n");

 strcpy(table[0][0],"");
 strcpy(table[0][1],"a");
 strcpy(table[0][2],"b");
 strcpy(table[0][3],"$");
 strcpy(table[1][0],"A");
 strcpy(table[2][0],"B");

 for(i=0;i<3;i++)
 {
  if(first[i][0]!='@')
   strcpy(table[numr(prod[i][0])][numr(first[i][0])],prod[i]);
  else
   strcpy(table[numr(prod[i][0])][numr(follow[i][0])],prod[i]);
 }
 printf("\n----------------------------------------------------------------\n");
 for(i=0;i<3;i++){
  for(j=0;j<4;j++)
  {
   printf("%-30s",table[i][j]);
   if(j==3) printf("\n----------------------------------------------------------------\n");
  }
 }

 printf("Enter the input string terminated with $ to parse:-");
 scanf("%s",input);
 for(i=0;input[i]!='\0';i++){
  if((input[i]!='a')&&(input[i]!='b')&&(input[i]!='$'))
  {
   printf("Invalid String");
   exit(0);
  }
 }
```

```
if(input[i-1]!='$')
 {
  printf("\n\nInput String Entered Without End Marker $\n");
  exit(0);
 }

push('$');
 push('A');
 i=0;

printf("\n\n");
 printf("Stack\t Input\tAction");
 printf("\n---------------------------------------------------------------------\n");

while(input[i]!='$'&&stack[top]!='$')
 {
  display();
  printf("\t\t%s\t",(input+i));
  if(stack[top]==input[i])
  {
   printf("\tMatched %c\n", input[i]);
   pop();
   i++;
  }
  else
  {
   if(stack[top]>=65&&stack[top]<92)
   {
    strcpy(curp,table[numr(stack[top])][numr(input[i])]);
    if(!(strcmp(curp,"e")))
    {
     printf("\nInvalid String - Rejected\n");
     exit(0);
    }
    else
    {
     printf("\tApply production %s\n",curp);
     if(curp[3]=='@')
     pop();
     else
     {
      pop();
      n=strlen(curp);
      for(j=n-1;j>=3;j--)
      push(curp[j]);
     }
    }
   }
  }
 }
```

```
  display();
  printf("\t\t%s\t",(input+i));
  printf("\n--------------------------------------------------------------------\n");

  if(stack[top]=='$'&&input[i]=='$')
  {
    printf("\nValid String - Accepted\n");
  }
  else
  {
    printf("Invalid String - Rejected\n");
  }
}
```



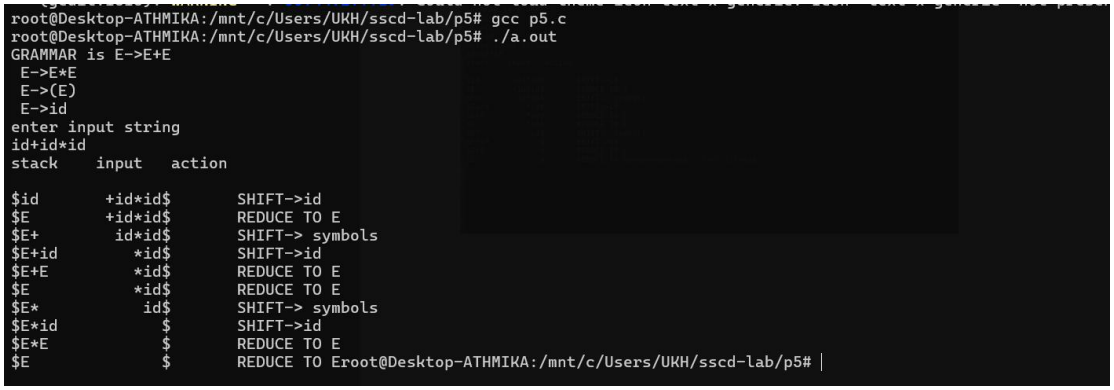
PROGRAM-5

5. Design, develop and implement YACC/C program to demonstrate Shift Reduce Parsing technique for the grammar rules: E ->E+T | T, T ->T*F | F, F ->(E) | id and parse the sentence: id + id * id.

P5.c

```
#include<stdio.h>
#include<string.h>
int k=0,z=0,i=0,j=0,c=0;
char a[16],ac[20],stk[15],act[10];

void check();

void main()
{
        puts("GRAMMAR is E->E+E \n E->E*E \n E->(E) \n E->id");
        puts("enter input string ");

        fgets(a, sizeof(a), stdin);
        a[strcspn(a, "\n")] = '\0';  // safely strip newline

        c=strlen(a);
        strcpy(act,"SHIFT->");

        puts("stack \t input \t action");

        for(i=0; j<c; i++,j++) {
                if(a[j]=='i' && a[j+1]=='d') {
                        stk[i]=a[j];
                        stk[i+1]=a[j+1];
                        stk[i+2]='\0';
                        a[j]=' ';
                        a[j+1]=' ';
                        printf("\n$%s\t%s$\t%sid",stk,a,act);

                        check();
                }
                else {
                        stk[i]=a[j];
                        stk[i+1]='\0';
                        a[j]=' ';
                        printf("\n$%s\t%s$\t%s symbols",stk,a,act);
                        check();
                }
        }
}
```

```
void check() {
	strcpy(ac,"REDUCE TO E");

	for(z=0; z<c; z++)
		if(stk[z]=='i' && stk[z+1]=='d')
		{
			stk[z]= 'E';
			stk[z+1]='\0';
			printf("\n$%s\t%s$\t%s",stk,a,ac);
			 j++;
		}

	for(z=0; z<c; z++)
		if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='E')
		{
		   stk[z]='E';
		   stk[z+1]='\0';
		   stk[z+2]='\0';
		   printf("\n$%s\t%s$\t%s",stk,a,ac);
		   i=i-2;
		}

	for(z=0; z<c; z++)
		if(stk[z]=='E' && stk[z+1]=='*' && stk[z+2]=='E')
		{
			stk[z]='E';
			stk[z+1]='\0';
			stk[z+2]='\0';
			printf("\n$%s\t%s$\t%s",stk,a,ac);
			   i=i-2;
		}

	for(z=0; z<c; z++)
		if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')')
		{
			stk[z]='E';
			stk[z+1]='\0';
			stk[z+2]='\0';
			printf("\n$%s\t%s$\t%s",stk,a,ac);
			   i=i-2;
		}
}
```


root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p5# gcc p5.c
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p5# ./a.out
GRAMMAR is E->E+E
E->E*E
E->(E)
E->id
enter input string
id+id*id
stack input action
$id +id*id$ SHIFT->id
$E +id*id$ REDUCE TO E
$E+ id*id$ SHIFT-> symbols
$E+id *id$ SHIFT->id
$E+E *id$ REDUCE TO E
$E *id$ REDUCE TO E
$E* id$ SHIFT-> symbols
$E*id $ SHIFT->id
$E*E $ REDUCE TO E
$E $ REDUCE TO Eroot@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p5#

PROGRAM-6

6.Design, develop and implement a C/Java program to generate the machine code using Triples for the statement A = -B * (C +D) whose intermediate code in three-address form:
T1 = -B
T2 = C + D
T3 = T1 * T2
A = T3

P6.c

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

char op[2],arg1[5],arg2[5],result[5];

void main()  {
        FILE *fp1,*fp2;

        fp1=fopen("input.txt","r+");
        fp2=fopen("output.txt","w+");

        while (fscanf(fp1,"%s%s%s%s",result,arg1,op,arg2) == 4) {

                if(strcmp(op,"+")==0) {
                        fprintf(fp2,"\nMOV R0,%s",arg1);
                        fprintf(fp2,"\nADD R0,%s",arg2);
                        fprintf(fp2,"\nMOV %s,R0",result);
                }
                if(strcmp(op,"*")==0) {
                        fprintf(fp2,"\nMOV R0,%s",arg1);
                        fprintf(fp2,"\nMUL R0,%s",arg2);
                        fprintf(fp2,"\nMOV %s,R0",result);
                }
                if(strcmp(op,"-")==0) {
                        fprintf(fp2,"\nMOV R0,%s",arg1);
                        fprintf(fp2,"\nSUB R0,%s",arg2);
                        fprintf(fp2,"\nMOV %s,R0",result);
                }
                if(strcmp(op,"/")==0) {
                        fprintf(fp2,"\nMOV R0,%s",arg1);
                        fprintf(fp2,"\nDIV R0,%s",arg2);
                        fprintf(fp2,"\nMOV %s,R0",result);
                }
                if(strcmp(op,"=")==0) {
```

```
                        fprintf(fp2,"\nMOV R0,%s",arg1);
                        fprintf(fp2,"\nMOV %s,R0",result);
                }
        }
        fclose(fp1);
        fclose(fp2);
}
```

Input.txt

```
T1 -B = ?
T2 C  +  D
T3 T1  *  T2
A T3 = ?
```

Output.txt

=>

```
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p6# gcc p6.c
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p6# ./a.out
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p6# cat input.txt
T1 -B = ?
T2 C  +  D
T3 T1  *  T2
A T3 = ?
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p6# cat output.txt

MOV R0,-B
MOV T1,R0
MOV R0,C
ADD R0,D
MOV T2,R0
MOV R0,T1
MUL R0,T2
MOV T3,R0
MOV R0,T3
MOV A,R0root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p6#
```

PROGRAM-7

7. Write a LEX program to eliminate comment lines in a C program and copy the resulting program into a separate file.

P7.l

```
%{
        #include<stdio.h>
        #include<stdlib.h>
        int comment=0;
        int state=1;
%}

%%

"//".*\n { comment++; }
"/*" { state=0; }
"*/" {
        if(state==0)
                comment++;
        state=1;
    }
.|\n {
        if(state==1)
                fprintf(yyout,"%s",yytext);
    }

%%

void main() {
        char inpfile[500],outfile[500];

        printf("enter input file name\n");
        scanf("%s",inpfile);

        printf("Enter output file name\n");
        scanf("%s",outfile);

        yyin=fopen(inpfile,"r");
        yyout=fopen(outfile,"w");

        yylex();
        printf("Total no. of comment line is: %d\n",comment);
}
```

File1.c

```
/*Adding two number  */
#include<stdio.h>
void main() {
    int a,b,sum;  //Declaring variables
    printf("Enter two number:"); /*asking the user to enter 2 no*/
    scanf("%d%d",&a,&b); //Taking input
    sum=a+b;              /*calculate sum*/

    printf("Result=%d",sum); //Printing the result
}
```

```
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p7# lex p7.l
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p7# cc lex.yy.c -ll
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p7# ./a.out
enter input file name
file1.c
Enter output file name
file2.c
Total no. of comment line is: 6
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p7# cat file2.c

 #include<stdio.h>
 void main()
 {
 int a,b,sum;   printf("Enter two number:");
 scanf("%d%d",&a,&b);  sum=a+b;

 printf("Result=%d",sum);  }
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p7#
```

PROGRAM-8

8. Write YACC program to recognize valid identifier, operators and keywords in the given text (C program) file.

P8.l

```
%{
        #include <stdio.h>
        #include "y.tab.h"
        extern int yylval;
%}
%%
[ \t] ;

[+|-|*|/|=|<|>] {
                printf("operator is %s\n",yytext);
                return OP;
        }

[0-9]+ {
                yylval = atoi(yytext);
                printf("numbers is %d\n",yylval);
                return DIGIT;
        }

int|char|bool|float|void|for|do|while|if|else|return|void {
                printf("keyword is %s\n",yytext);
                return KEY;
        }

[a-zA-Z0-9]+ {
                printf("identifier is %s\n",yytext);
                return ID;
        }

. ;
%%
```

P8.y

```
%{
        #include <stdio.h>
        #include <stdlib.h>
        int yylex();
        void yyerror(const char *);
        extern int yylval;
        int id=0, dig=0, key=0, op=0;
%}
```

```
%token DIGIT ID KEY OP
%%
input: DIGIT input { dig++; }
        | ID input { id++; }
        | KEY input { key++; }
        | OP input {op++;}
        | DIGIT { dig++; }
        | ID { id++; }
        | KEY { key++; }
        | OP { op++;}
        ;
%%
#include <stdio.h>
extern int yylex();
extern int yyparse();
extern FILE *yyin;

int main() {
        FILE *myfile = fopen("p8.c", "r");
        if (!myfile) {
                printf("I can't open sam_input.c!");
                return -1;
        }
        yyin = myfile;
        do {
                yyparse();
        } while (!feof(yyin));
        printf("numbers = %d\nKeywords = %d\nIdentifiers = %d\noperators = %d\n",
dig, key,id, op);
}

void yyerror(const char *s) {
        printf("EEK, parse error! Message: %s\n", s);
        exit(-1);
}
```

P8.c

```
void main(){
        int a;
        float bc;
        char c;
        char ch;
        if(a == 80)
                printf(“Good”);
        else
                printf(“Bad”);
}
```

```
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p8# lex p8.l
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p8# yacc -d p8.y
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p8# cc lex.yy.c y.tab.c -ll
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p8# ./a.out
keyword is void
identifier is main

keyword is int
identifier is a

keyword is float
identifier is bc

keyword is char
identifier is c

keyword is char
identifier is ch

keyword is if
identifier is a
operator is =
operator is =
numbers is 80

identifier is printf
identifier is Good

keyword is else

identifier is printf
identifier is Bad


numbers = 1
Keywords = 7
Identifiers = 10
operators = 2
root@Desktop-ATHMIKA:/mnt/c/Users/UKH/sscd-lab/p8#
```