Experiment No: 1a USN: Date:

/*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 and operators present and print them separately. */

```
%{
#include<stdio.h>
int op=0,oc=0;
%}

%%
[+*] {oc++;}
[0-9] {op++;}
%%

int main()
{
printf("enter arithmetic expression\n");
yylex();
if(op==oc+1)
printf("valid arithmetic expression\n");
else
printf("invalid expression");
printf("operators=%d\n",oc);
printf("operands=%d\n",op);
}

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

Experiment No: 1a USN:

OUTPUT:

```
[student@localhost ~]$ lex arith.l
[student@localhost ~]$ cc lex.yy.c
[student@localhost ~]$ ./a.out
enter arithmetic expression
0+1

valid arithmetic expression
operators=1
operands=2

[student@localhost ~]$ ./a.out
enter arithmetic expression
a**b
ab
invalid expressionoperators=2
operands=0

[student@localhost ~]$ ./a.out
enter arithmetic expression
2*5+3

valid arithmetic expression
operators=2
operands=3

[student@localhost ~]$ ./a.out
enter arithmetic expression
2++3

invalid expressionoperators=2
operands=2
```

/* Write YACC program to Evaluate arithmetic expression involving
Operators + , - , * and / */

Lex Program:

```
%{
#include"y.tab.h"
extern int yylval;
%}

%%
[0-9] {yylval=atoi(yytext); return NUM;}
. {return yytext[0];}
%%

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

YACC Program:

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

%token NUM
%left '+' '-'
%right '*' '/'
%%
start: exp { printf("RES=%d",$1);}
     ;
exp: exp '+' exp {$$=$1+$3;}
   | exp '-' exp {$$=$1-$3;}
   | exp '*' exp {$$=$1*$3;}
   | exp '/' exp {if ($3==0){printf("divide by zero\n");exit(0);}
      else{$$=$1/$3;}}
   | NUM {$$=$1;}
;
%%
```

```
main()
{
printf("enter expression");
yyparse();
printf("valid expression");
}

yyerror()
{
printf("invalid expression");
exit(0);
}
```

OUTPUT:

```
[student@localhost ~]$ yacc -d 1b.y
[student@localhost ~]$ lex 1b.l
[student@localhost ~]$ cc y.tab.c lex.yy.c
[student@localhost ~]$ ./a.out
enter expression2+3
RES=5valid expression

[student@localhost ~]$ ./a.out
enter expression4-6+3
RES=1valid expression

[student@localhost ~]$ ./a.out
enter expression2+5-6
RES=1valid expression

[student@localhost ~]$ ./a.out
enter expression6-*7
invalid expression
```

Experiment No: 2 USN: Date:

/* Develop Implement and execute a program using YACC tool to recognize all strings ending with *b* preceded by *n a's* using the grammar $a^n b$ (note: input n value) */

LEX Program:

```
%{
#include "y.tab.h"
%}

%%
[Aa] {return A;}
[Bb] {return B;}
.;
%%
int yywrap()
{
return 1;
}
```

YACC Program:

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

%token A B

%%
st:A s B
  |B
  ;
s:A s
 |
 ;
%%
main()
{
printf("enter string:");
yyparse();
printf("valid expression");
}
yyerror()
{
printf("invalid string");
exit(0);
}
```

Experiment No: 2 USN:

OUTPUT:

```
[student@localhost ~]$ yacc -d yac1.y
[student@localhost ~]$ lex yac1.l
[student@localhost ~]$ cc lex.yy.c y.tab.c
[student@localhost ~]$ ./a.out
enter string:aaab

valid expression

[student@localhost ~]$ ./a.out
enter string:abbb
invalid string

[student@localhost ~]$ ./a.out
enter string:b

valid expression

[student@localhost ~]$ ./a.out
enter string:aaaabbbb
invalid string
```

Experiment No: 3 USN: Date:

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

```
#include<stdio.h>
#include<string.h>
void main()
{
char fin[10][20],st[10][20],ft[20][20],fol[20][20];
int a=0,e,i,t,b,c,n,k,l=0,j,s,m,p;
printf("enter the no. of coordinates\n");
scanf("%d",&n);
printf("enter the productions in a grammar\n");
for(i=0;i<n;i++)
scanf("%s",st[i]);
for(i=0;i<n;i++)
fol[i][0]='\0';
for(s=0;s<n;s++)
{
for(i=0;i<n;i++)
{
j=3;
l=0;
a=0;
l1:if(!((st[i][j]>64)&&(st[i][j]<91)))
{
for(m=0;m<l;m++)
{
if(ft[i][m]==st[i][j])
goto s1;
}
ft[i][l]=st[i][j];
l=l+1;
s1:j=j+1;
}
else
{
if(s>0)
{
while(st[i][j]!=st[a][0])
{
a++;
}
b=0;
while(ft[a][b]!='\0')
{
for(m=0;m<l;m++)
{
if(ft[i][m]==ft[a][b])
goto s2;
}
```

```
ft[i][l]=ft[a][b];
l=l+1;
s2:b=b+1;
}
}
}
while(st[i][j]!='\0')
{
if(st[i][j]=='|')
{
j=j+1;
goto l1;
}
j=j+1;
}
ft[i][l]='\0';
}
}
printf("first pos\n");
for(i=0;i<n;i++)
printf("FIRS[%c]=%s\n",st[i][0],ft[i]);
fol[0][0]='$';
for(i=0;i<n;i++)
{
k=0;
j=3;
if(i==0)
l=1;
else
l=0;
k1:while((st[i][0]!=st[k][j])&&(k<n))
{
if(st[k][j]=='\0')
{
k++;
j=2;
}
j++;
}
j=j+1;
if(st[i][0]==st[k][j-1])
{
if((st[k][j]!='|')&&(st[k][j]!='\0'))
{
a=0;
if(!((st[k][j]>64)&&(st[k][j]<91)))
{
for(m=0;m<l;m++)
{
if(fol[i][m]==st[k][j])
```

```
goto q3;
}
q3:
fol[i][l]=st[k][j];
l++;
}
else
{
while(st[k][j]!=st[a][0])
{
a++;
}
p=0;
while(ft[a][p]!='\0')
{
if(ft[a][p]!='@')
{
for(m=0;m<l;m++)
{
if(fol[i][m]==ft[a][p])
goto q2;
}
fol[i][l]=ft[a][p];
l=l+1;
}
else
e=1;
q2:p++;
}
if(e==1)
{
e=0;
goto a1;
}
}
}
else
{
a1:c=0;
a=0;
while(st[k][0]!=st[a][0])
{
a++;
}
while((fol[a][c]!='\0')&&(st[a][0]!=st[i][0]))
{
for(m=0;m<l;m++)
{
if(fol[i][m]==fol[a][c])
goto q1;
```

```
goto q3;
}
q3:
fol[i][l]=st[k][j];
l++;
}
else
{
while(st[k][j]!=st[a][0])
{
a++;
}
p=0;
while(ft[a][p]!='\0')
{
if(ft[a][p]!='@')
{
for(m=0;m<l;m++)
{
if(fol[i][m]==ft[a][p])
goto q2;
}
fol[i][l]=ft[a][p];
l=l+1;
}
else
e=1;
q2:p++;
}
if(e==1)
{
e=0;
goto a1;
}
}
}
else
{
a1:c=0;
a=0;
while(st[k][0]!=st[a][0])
{
a++;
}
while((fol[a][c]!='\0')&&(st[a][0]!=st[i][0]))
{
for(m=0;m<l;m++)
{
if(fol[i][m]==fol[a][c])
goto q1;
```

```
{
b=0;
a=0;
while(st[a][0]!=st[i][3])
{
a++;
}
while(ft[a][b]!='\0')
{
printf("M[%c,%c]=%s\n",st[i][0],ft[a][b],fin[s]);
b++;
}
}
s++;
}
if(st[i][j]=='|')
j++;
}
}
}
```

Experiment No: 3 USN: Date:

OUTPUT:

```
[student@localhost ~]$ ./a.out
enter the no. of coordinates
2
enter the productions in a grammar
A->aBa
B->bB|@
first pos
FIRS[A]=a
FIRS[B]=b@
follow pos
FOLLOW[A]=$
FOLLOW[B]=a

M[A,a]=A->aBa
M[B,b]=B->bB
M[B,a]=B->@
```

Experiment No: 4 USN: Date:

/* 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*** */

```
#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 ");
gets(a);
c=strlen(a);
strcpy(act,"SHIFT->");
puts("stack \t input \t action");
for(k=0,i=0; j<c; k++,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%ssymbols",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+1]='\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+1]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);
i=i-2;
}
}
```

OUTPUT:

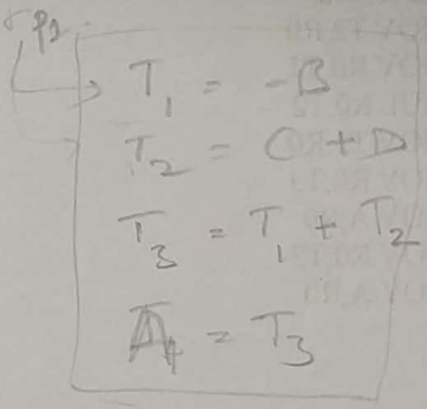
```
[student@localhost ~]$ cc lab4.c
[student@localhost ~]$ ./a.out
GRAMMAR is E->E+E
 E->E*E
 E->(E)
 E->id
enter input string
id+id*id
stack    input   action

$id      +id*id$      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 E
```

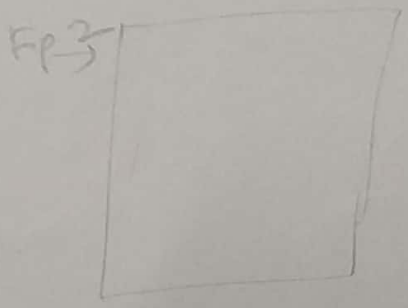
Experiment No: 5 USN: Date:

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

```
#include<stdio.h>
#include<stdlib.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(!feof(fp1))
{
fscanf(fp1,"%s%s%s%s",result,arg1,op,arg2);
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);
}
```





OUTPUT:

```
[student@localhost ~]$ cc lab5.c
[student@localhost ~]$ ./a.out
[student@localhost ~]$ 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,R0
MOV R0,T3
MOV A,R0
```

Experiment No: 6a USN: Date:

/* Write a LEX Program to eliminate comment lines in a C program and copy the resulting program in to a separate file. */

```
%{
#include<stdio.h>
int cc=0;
FILE *sp;
%}
%%
[/][/].*[\n] {cc++; fprintf(sp," ",yytext);}
[/][*].*[*][/] {cc++; fprintf(sp," ",yytext);}
.|\n {fprintf(sp,"%s",yytext);}
%%
main(int argc,char **argv)
{
FILE *f1;
if(argc>1)
{
f1=fopen(argv[1],"r");
sp=fopen(argv[2],"w");
if(!f1)
{
printf("couldn't open the file\n");
exit(1);
}
yyin=f1;
}
yylex();
printf("numbers of comment lines=%d\n",cc);
}
int yywrap()
{
return 1;
}
```

Experiment No: 6a USN:

OUTPUT:

```
[student@localhost ~]$ lex  lab1.l
[student@localhost ~]$ cc lex.yy.c
[student@localhost ~]$ ./a.out c.c out.c
numbers of comment lines=3
[student@localhost ~]$ cat c.c
#include<stdio.h>
main() //main starts
{
 int a,b,c;  //declarations
 a=10;
b=5;
c=a+b;
/*print the result*/
printf("a+b=%d\n",c);
}


[student@localhost ~]$ cat out.c
#include<stdio.h>
main()  {
 int a,b,c;    a=10;
b=5;
c=a+b;

printf("a+b=%d\n",c);
}
[student@localhost ~]$
```