DAYANANDA SAGAR COLLEGE OF ENGINEERING

SHAVIGE MALLESHWARA HILLS, K.S. LAYOUT, BANGALORE -560078.

DEPARTMENT
OF
COMPUTER SCIENCE AND ENGINEERING

System Software & Operating System Laboratory

Subject Code: 18CS6DLSSL

Semester: VI

Vision and Mission of the Department

Vision

To provide a vibrant learning environment in computer science and engineering with focus on industry needs and research, for the students to be successful global professionals contributing to the society.

Mission

- To adopt a contemporary teaching learning process with emphasis on hands on and collaborative learning.
- To facilitate skill development through additional training and encourage student forums for enhanced learning.
- To collaborate with industry partners and professional societies and make the student industry ready.
- To encourage innovation through multidisciplinary research and development activities.
- To inculcate human values and ethics in students and groom them to be responsible citizens.

Do's and Don'ts

Do's

1. Read and understand the program thoroughly before coming to the laboratory.
2. Report any broken plugs or exposed electrical wires to your faculty/laboratory technician immediately.
3. Turn off the machine once you have finished using it.
4. Maintain silence while working in the lab.
5. Keep the Computer lab premises clean and tidy.
6. Place backpacks under the table or computer counters.

Don'ts

1. Do not talk on cell phones in the lab.
2. Do not eat or drink in the laboratory.
3. Do not touch, connect or disconnect any plug or cable without the faculty/laboratory technician's permission.
4. Do not misbehave in the computer laboratory.
5. Do not install or download any software or modify or delete any system files on any lab computers.
6. Do not read or modify other users' files.
7. Do not plug in external devices without scanning them for computer viruses.
8. Please treat fellow users of the laboratory, and all equipment within the laboratory, with the appropriate level of care and respect.
9. Do not leave your personal belongings unattended. We are not responsible for any theft.

Head,
Dept. of CSE

1. Install Linux Operating System and explore the Linux System Environment

2. Explore various Linux Internal and External Commands

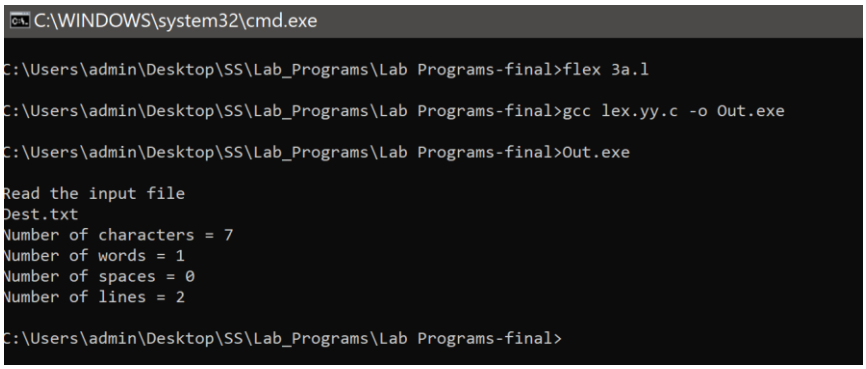
3. a) Program to count the number of characters, words, spaces and lines from a given input file.

```
%{
#include <stdio.h>
int wc=0,cc=0,lc=0,sc=0;
char infile[25];
%}

word [a-zA-Z0-9]+
eol [\n]

%%
{word}      {wc++; cc+=yyleng;}
{eol} {lc++; cc++;}
[" "] {sc++; cc++;}
[\t]  {sc+=8; cc++;}
.     {cc++;}
%%
int yywrap()
{ }
int main()
{
      printf("\nRead the input file\n");
      scanf("%s",infile);
      yyin=fopen(infile,"r");
      yylex();
      printf("Number of characters = %d\n",cc);
      printf("Number of words = %d\n",wc);
      printf("Number of spaces = %d\n",sc);
      printf("Number of lines = %d\n",lc);
      return 0;
      fclose(yyin);

}
```

Command for execution:

```
lex pgm_name.l

OR

Flex pgm.l

gcc lex.yy.c -o pgm_name.exe

pgm_name.exe
```

output:



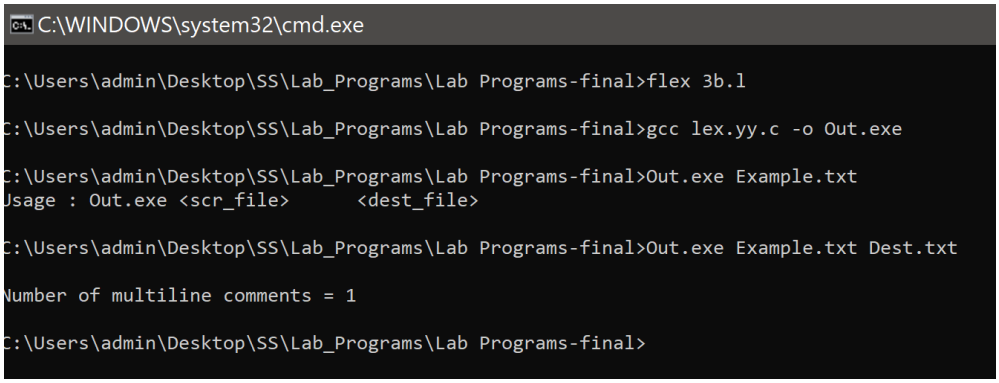
b) Program to count the number of comment lines in a given C Program. Also eliminate them and copy it to a separate file.

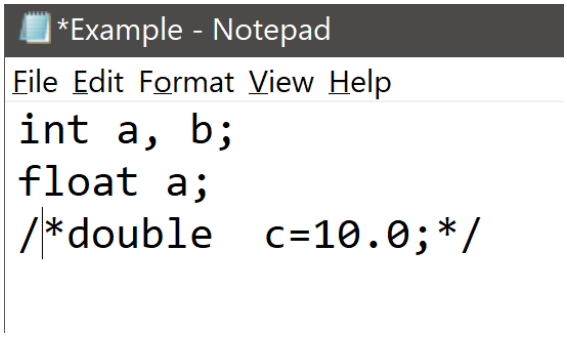
```
%{
#include <stdio.h>
int cc=0;
%}

%x CMNT

%%
"/*" {BEGIN CMNT;}
<CMNT>. ;
<CMNT>"*/" {BEGIN 0; cc++;}
%%

int yywrap()  { }

int main(int argc, char *argv[])
{
      if(argc!=3)
      {
            printf("Usage : %s    <scr_file> <dest_file>\n",argv[0]);
            return 0;
      }
      yyin=fopen(argv[1],"r");
      yyout=fopen(argv[2],"w");
      yylex();
      printf("\nNumber of multiline comments = %d\n",cc);
      return 0;
}
```

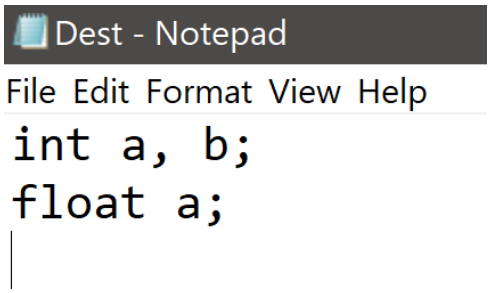
Command for execution:

```
lex pgm_name.l

OR

Flex pgm.l
```

```
gcc lex.yy.c -o pgm_name.exe
pgm_name.exe
```

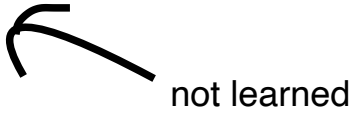






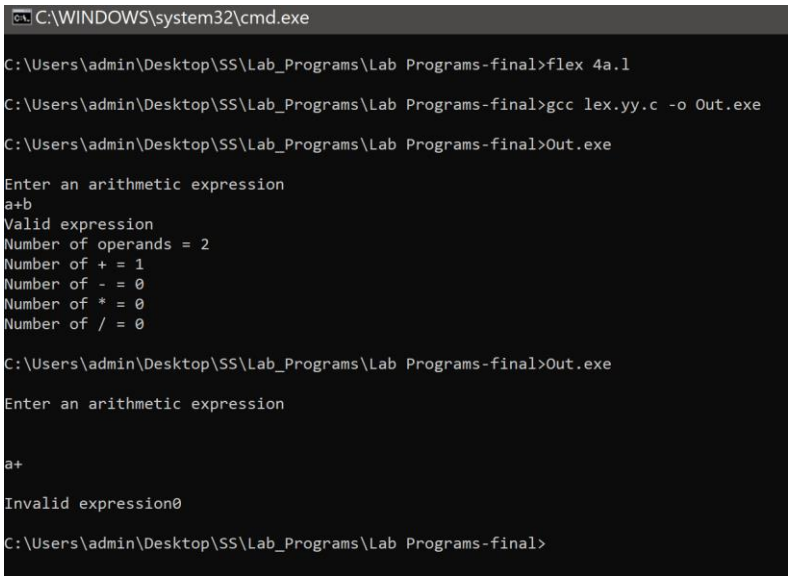
4. a) Lex program to recognize a valid arithmetic expression and to recognize identifiers and operators present and print them separately

```
%{

#include <stdio.h>
int ext(int);
int a[]={0,0,0,0},valid=1,opnd=0,top=-1,i;

%}

%x oper

%%
["("]                    {top++;}
[")"]                    {top--;}
[a-zA-Z0-9]+             {BEGIN oper; opnd++;}
<oper>"+"        {if(valid) {valid = 0; i = 0;} else ext(0);}
<oper>"-"        {if(valid) {valid = 0; i = 1;} else ext(0);}
<oper>"*"        {if(valid) {valid = 0; i = 2;} else ext(0);}
<oper>"/"        {if(valid) {valid = 0; i = 3;} else ext(0);}
<oper>"("        {top++;}
<oper>")"        {top--;}
<oper>[a-zA-Z0-9]+    {opnd++; if(valid == 0) {valid = 1; a[i]++;} else
ext(0);}
<oper>"\n"       {if(valid == 1 && top == -1) {printf("Valid
expression\n"); return 0;} else ext(0);}
.\n              ext(0);
%%

int yywrap() {  }

int ext(int x)
{
      printf("\nInvalid expression%d\n",x);
      exit(0);
}

int main()
{
      printf("\nEnter an arithmetic expression\n");
      yylex();
      printf("Number of operands = %d\n",opnd);
      printf("Number of + = %d\n",a[0]);
      printf("Number of - = %d\n",a[1]);
      printf("Number of * = %d\n",a[2]);
      printf("Number of / = %d\n",a[3]);
      return 0;
}
```



Command for execution:

```
lex pgm_name.l
OR
Flex pgm.l
gcc lex.yy.c -o pgm_name.exe
pgm_name.exe
```



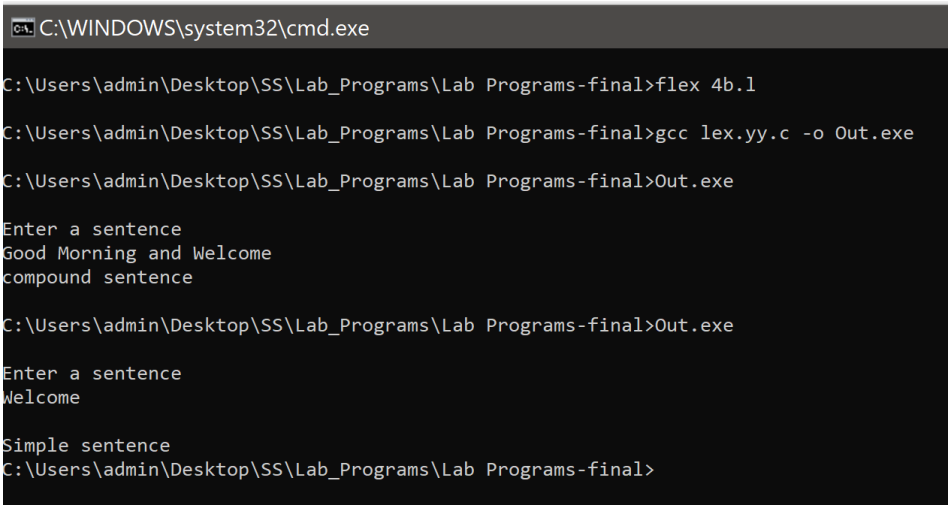
b) Program to check whether a given sentence is simple or compound

```
%{
#include<stdio.h>
%}
ws [ \t\n]
%%
{ws}"and"{ws}|{ws}"AND"{ws} |
{ws}"or"{ws}|{ws}"OR"{ws} |
{ws}"but"{ws}|{ws}"BUT"{ws} |
{ws}"because"{ws} |
{ws}"nevertheless"{ws}      {printf("compound sentence\n");exit(0);}
.     ;
\n    return 0;
%%

int yywrap() {   }
int main()
{
      printf("\nEnter a sentence\n");
      yylex();
      printf("Simple sentence");
      exit(0);
      //return 0;
```

```
}
```

Command for execution:

```
lex pgm_name.l
OR
Flex pgm.l
gcc lex.yy.c -o pgm_name.exe
pgm_name.exe
```

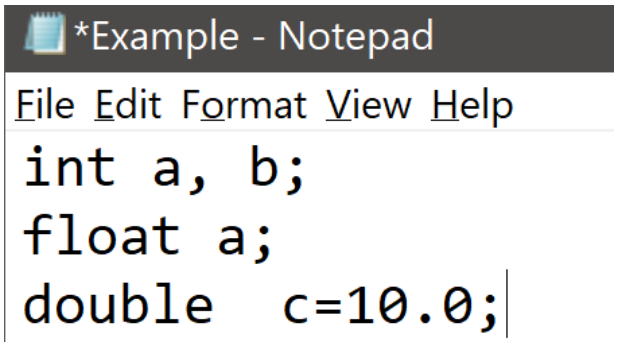


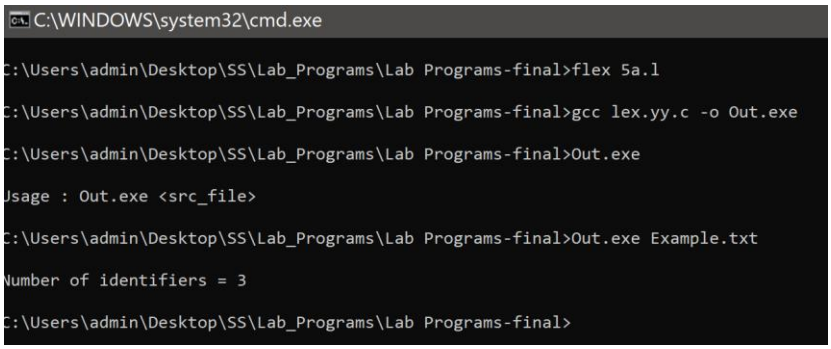
5. a) Program to recognize and count the number of identifiers in a given input file

```
%{
#include<stdio.h>
int idc=0;
%}

e[=][   ]*[0-9]+
ws[   \n\t]*
id[_a-zA-Z][_a-zA-Z0-9]*
decln "int"|"float"|"clear"|"double"|"short"|"long"|"unsigned"

%x defn

%%
{decln}                     {BEGIN defn;}
<defn>{ws}{id}{ws}\,        {idc++;}
<defn>{ws}{id}{ws}\;        {BEGIN 0;idc++;}
<defn>{ws}{id}{ws}{e}{ws}\, {idc++;}
<defn>{ws}{id}{ws}{e}{ws}\; {BEGIN 0;idc++;}
<*>\n       ;
<*>.        ;
%%

int yywrap() {   }

int main(int argc,char *argv[])
{
      if(argc==2)
      {
            yyin=fopen(argv[1],"r");
            yylex();
            printf("\nNumber of identifiers = %d\n",idc);
      }
      else
      {
            printf("\nUsage : %s  <src_file> \n",argv[0]);
      }
      return 0;
}
```

Command for execution:

```
lex pgm_name.l

OR

Flex pgm.l

gcc lex.yy.c -o pgm_name.exe

pgm_name.exe
```





b) Program to evaluate an arithmetic expression involving operators +,-,*,/

5b.l

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

5b.y

```
%{#include <stdio.h>%}
%token NUM
%left '+''-'
```

```
%left '/''*'
%%
expr:e {printf("Valid expression\n"); printf("Result : %d\n",$1);
return 0;}
e:e'+'e {$$=$1+$3;}
| e'-'e {$$=$1-$3;}
| e'*'e {$$=$1*$3;}
| e'/'e {$$=$1/$3;}
| '('e')' {$$=$2;}
| NUM {$$=$1;}
%%
int main()
{
printf("\nEnter an arithmetic expression\n");
yyparse();
return 0;}
int yyerror()
{
printf("\nInvalid expression\n");
return 0;
}
```

Command for execution:

```
lex pgm_name.l
OR
Flex pgm.l
Yacc -y -d pgm_name.y
OR
Bison -y -d pgm_name.y
gcc lex.yy.c y.tab.c -o pgm_name.exe
pgm_name.exe
```

```
C:\WINDOWS\system32\cmd.exe

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>flex 5b.l

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>bison -y -d 5b.y

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>gcc lex.yy.c y.tab.c -o Out.exe
y.tab.c: In function 'yyparse':
y.tab.c:583:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
 # define YYLEX yylex ()
                ^
y.tab.c:1228:16: note: in expansion of macro 'YYLEX'
       yychar = YYLEX;
                ^~~~
y.tab.c:1391:7: warning: implicit declaration of function 'yyerror' [-Wimplicit-function-declaration]
       yyerror (YY_("syntax error"));
       ^~~~~~~

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>Out.exe

Enter an arithmetic expression
2+2
Valid expression
Result : 4

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>Out.exe

Enter an arithmetic expression
6/3
Valid expression
Result : 2

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>Out.exe

Enter an arithmetic expression
5*

Invalid expression

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>
```

6. a) Program to recognize a valid arithmetic expression that uses operates +,-,*,/

6a.l

```
%{
#include "y.tab.h"
%}
%%
[0-9]+(\.[0-9]+)? {return NUM;}
[a-zA-Z][_a-zA-Z0-9]* {return ID;}
[\t] ;
\n {return 0;}
. {return yytext[0];}
%%
int yywrap() { }
```

6a.y

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

%token L D NL

%%
var: L E NL {printf("Valid Variable\n");return 0;}
E: E L
| E D
| ;
%%

int yyerror()
{
printf("\n Invalid Variable\n");
```

wrong hai

```
return 0;
}

int main()
{
printf("\nEnter a variable\n");
yyparse();
}
```

Command for execution:

```
lex pgm_name.l
OR
Flex pgm.l
Yacc -y -d pgm_name.y
OR
Bison -y -d pgm_name.y
gcc lex.yy.c y.tab.c -o pgm_name.exe
pgm_name.exe
```

```
C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>flex 6a.l

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>bison -y -d 6a.y

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>gcc lex.yy.c y.tab.c -o out.exe
y.tab.c: In function 'yyparse':
y.tab.c:588:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
 # define YYLEX yylex ()
                ^
y.tab.c:1233:16: note: in expansion of macro 'YYLEX'
       yychar = YYLEX;
                ^~~~~
6a.y:8:43: warning: implicit declaration of function 'exit' [-Wimplicit-function-declaration]
 expr:e { printf("This is Valid Expression"); exit(0);}
                                           ^~~~
6a.y:8:43: warning: incompatible implicit declaration of built-in function 'exit'
6a.y:8:43: note: include '<stdlib.h>' or provide a declaration of 'exit'
y.tab.c:1354:7: warning: implicit declaration of function 'yyerror' [-Wimplicit-function-declaration]
       yyerror (YY_("syntax error"));
       ^~~~~~~
6a.y: In function 'yyerror':
6a.y:27:1: warning: incompatible implicit declaration of built-in function 'exit'
 exit(0);
 ^~~~
6a.y:27:1: note: include '<stdlib.h>' or provide a declaration of 'exit'

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>Out.exe

Type the expression to be tested
a+b
This is Valid Expression
C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>Out.exe

Type the expression to be tested
a-b
This is Valid Expression
C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>Out.exe

Type the expression to be tested
a*

 Invalid expression!

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>
```

b) Program to recognize a valid variable, which starts with a letter, followed by any number of letters or digits

6b.l

```
%{

#include "y.tab.h"

%}

%%

[a-z] return L;

[0-9] return D;

\n {return NL;}
```

```
%%
int yywrap() { }
```

6b.y

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

%token L D NL

%%
var: L E NL {printf("Valid Variable\n");return 0;}
E: E L
| E D
| ;
%%

int yyerror()
{
printf("\n Invalid Variable\n");

return 0;
}

int main()
{
printf("\nEnter a variable\n");
yyparse();
}
```

Command for execution:

```
lex pgm_name.l
```

OR

```
Flex pgm.l
Yacc -y -d pgm_name.y
```

OR

```
Bison -y -d pgm_name.y
gcc lex.yy.c y.tab.c -o pgm_name.exe
pgm_name.exe
```

```
C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>flex 6b.l

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>bison -y -d 6b.y

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>gcc lex.yy.c y.tab.c -o Out.exe
y.tab.c: In function 'yyparse':
y.tab.c:583:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
 # define YYLEX yylex ()
                ^
y.tab.c:1228:16: note: in expansion of macro 'YYLEX'
       yychar = YYLEX;
                ^~~~
y.tab.c:1349:7: warning: implicit declaration of function 'yyerror' [-Wimplicit-function-declaration]
       yyerror (YY_("syntax error"));
       ^~~~~~

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>Out.exe

Enter a variable
a1
Valid Variable

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>Out.exe

Enter a variable
aa4
Valid Variable

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>Out.exe

Enter a variable
1a

 Invalid Variable

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>
```

7. a) Program to recognize the grammar ($a^n b$, n>=10)

7a.l

```
%{
#include "y.tab.h"
%}
%%
[aA] {return A;}
[bB] {return B;}
\n {return NL;}
. {return yytext[0];}
%%
int yywrap() { }
```

7a.y

```
%{
#include<stdio.h>
#include<stdlib.h>
%}
%token A B NL

%%
stmt: A A A A A A A A A S B NL {printf("valid string\n"); exit(0);}
;
S: S A
|

;
%%
int yyerror(char *msg)
{
printf("invalid string\n");
exit(0);
}
main()
```

```
{
printf("enter the string\n");
yyparse();
}
```

Command for execution:

```
lex pgm_name.l
OR
Flex pgm.l
Yacc -y -d pgm_name.y
OR
Bison -y -d pgm_name.y
gcc lex.yy.c y.tab.c -o pgm_name.exe
pgm_name.exe
```

```
C:\WINDOWS\system32\cmd.exe

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>flex 7a.l

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>bison -y -d 7a.y

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>gcc lex.yy.c y.tab.c -o Out.exe
y.tab.c: In function 'yyparse':
y.tab.c:589:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
 # define YYLEX yylex ()
                ^
y.tab.c:1234:16: note: in expansion of macro 'YYLEX'
       yychar = YYLEX;
                ^~~~~
y.tab.c:1355:7: warning: implicit declaration of function 'yyerror' [-Wimplicit-function-declaration]
       yyerror (YY_("syntax error"));
       ^~~~~~~
7a.y: At top level:
7a.y:20:1: warning: return type defaults to 'int' [-Wimplicit-int]
 main()
 ^~~~

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>Out.exe
enter the string
aaaaab
invalid string

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>Out.exe
enter the string
aaaaaaaaaab
valid string

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>Out.exe
enter the string
aaaaaaaaaaaaaab
valid string

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>
```

b) Program to recognize strings aaab,abbb,ab and a using the grammar ($a^n b^n, n>=0$)

7b.l

```
%{
#include "y.tab.h"
%}
%%
[aA] {return A;}
[bB] {return B;}
\n {return NL;}
. {return yytext[0];}
%%
int yywrap() { }
```

7b.y

```
%{
#include<stdio.h>
#include<stdlib.h>
%}
%token A B NL

%%
stmt: S NL { printf("valid string\n"); exit(0); }
;
S: A S B |
;
%%
int yyerror()
{
printf("invalid string\n");
exit(0);
}
main()
{
printf("enter the string\n");
yyparse();
}
```

Command for execution:

```
lex pgm_name.l
OR
Flex pgm.l
Yacc -y -d pgm_name.y
OR
Bison -y -d pgm_name.y
gcc lex.yy.c y.tab.c -o pgm_name.exe
pgm_name.exe
```

```
C:\WINDOWS\system32\cmd.exe

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>flex 7b.l

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>bison -y -d 7b.y

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>gcc lex.yy.c y.tab.c -o Out.exe
y.tab.c: In function 'yyparse':
y.tab.c:584:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
 # define YYLEX yylex ()
                ^
y.tab.c:1229:16: note: in expansion of macro 'YYLEX'
       yychar = YYLEX;
                ^~~~
y.tab.c:1350:7: warning: implicit declaration of function 'yyerror' [-Wimplicit-function-declaration]
       yyerror (YY_("syntax error"));
       ^~~~~~
7b.y: At top level:
7b.y:19:1: warning: return type defaults to 'int' [-Wimplicit-int]
 main()
 ^~~~

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>Out.exe
enter the string
aab
invalid string

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>Out.exe
enter the string
aabb
valid string

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>Out.exe
enter the string
ab
valid string

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>Out.exe
enter the string

valid string

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>
```

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

```
#include<stdio.h>
#include<conio.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();
}
}
getch();
}
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;
}
}
```

Command for execution:

```
gcc pgm_name.c -o pgm_name.exe

pgm_name.exe
```

```
C:\WINDOWS\system32\cmd.exe - Out.exe

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>gcc 8.c -o Out.exe

C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>Out.exe
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
C:\Users\admin\Desktop\SS\Lab_Programs\Lab Programs-final>Out.exe
GRAMMAR is E->E+E
 E->E*E
 E->(E)
 E->id
enter input string
id+id*
stack    input   action

$id       +id*$ SHIFT->id
$E        +id*$ REDUCE TO E
$E+        id*$ SHIFT->symbols
$E+id        *$ SHIFT->id
$E+E         *$ REDUCE TO E
$E           *$ REDUCE TO E
$E*           $ SHIFT->symbols
```

9. a) Write a $hell script that accepts two file names as arguments, checks if the permissions for both the files are identical and if the permissions are identical, outputs the common permissions; otherwise outputs each file name followed by its permissions.

```
if [ $# != 2 ]
then
echo  "Invalid input!!!"
else
p1=`ls -l $1|cut -d " " -f1`
p2=`ls -l $2|cut -d " " -f1`
if [ $p1 == $p2 ]
then
echo "the file permissions are same and it is : "
echo "$p1"
else
echo "The file permissions are different"
echo "$1 : $p1"
echo "$2 : $p2"
fi
fi
```

Command for execution:

```
Sh 9a.sh filename1 filename2
```

```
[root@localhost ~]# ls
9a.sh  bench.py  hello.c
[root@localhost ~]# ls -l
total 12
-rw------- 1 1000 root 277 Aug  2 17:39 9a.sh
-rw-r--r-- 1 root root 114 Dec 26  2020 bench.py
-rw-r--r-- 1 root root 185 Sep  9  2018 hello.c
[root@localhost ~]# sh 9a.sh
Invalid input!!!
[root@localhost ~]# sh 9a.sh 9a.sh bench.py
The file permissions are different
9a.sh : -rw-------
bench.py : -rw-r--r--
[root@localhost ~]# sh 9a.sh bench.py hello.c
the file permissions are same and it is :
-rw-r--r--
[root@localhost ~]#
```

b) Write a C program that creates a child process to read commands from the standard input and execute them

```
#include<stdio.h>
int main()
{
	int ch,rv;
	char cmd[10];
	rv=fork();
	if(rv==0)
	{
		do
		{
			printf("\nEnter a command\n");
			scanf("%s",cmd);
			system(cmd);
			printf("\n1 : continue\n0 : exit\n");
			scanf("%d",&ch);
		}
		while(ch!=0);
	}
	else
	{
		wait(0);
		printf("\nChild terminated\n");
	}
return 0;
}
```

Command for execution:

```
cc pgm_name.c
./a.out
```

```
[root@localhost ~]# cc 9b.c
9b.c: In function ‘main’:
9b.c:6:5: warning: implicit declaration of function ‘fork’ [-Wimplicit-function-
declaration]
    6 |  rv=fork();
      |     ^~~~
9b.c:13:4: warning: implicit declaration of function ‘system’ [-Wimplicit-functi
on-declaration]
   13 |    system(cmd);
      |    ^~~~~~
9b.c:21:3: warning: implicit declaration of function ‘wait’ [-Wimplicit-function
-declaration]
   21 |   wait(0);
      |   ^~~~
[root@localhost ~]# ./a.out

Enter a command
cal
    August 2021
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31


1 : continue
0 : exit
1
```

```
1

Enter a command
date
Mon Aug  2 05:43:04 PM UTC 2021

1 : continue
0 : exit
0

Child terminated
[root@localhost ~]#
```

10. a) Write a C/Java program that creates a zombie and then calls system to execute the ps command to verify that the process is zombie.

```
#include<stdio.h>
        #include <stdlib.h>
        #include <sys/types.h>
        #include <unistd.h>
        int main()
        {
                pid_t child_pid;
                                        /* Create a child process. */
                child_pid = fork ();
                if (child_pid > 0)
                {
                        printf("This is the parent process: %d. Sleep
for a minute\n",getpid());
                        sleep (60);
                }
                else
                {
                        printf("This is the child process: %d. Exit
immediately\n",getpid());
                        exit (0);
                }
                        system("ps -e -o pid,ppid,stat,comm");
                       return 0;
        }
```

Command for execution:

```
cc pgm_name.c
./a.out
```

```
[root@localhost ~]# cc 10a.c
[root@localhost ~]# ./a.out
This is the parent process: 111. Sleep for a minute
This is the child process: 112. Exit immediately
  PID  PPID STAT COMMAND
    1     0 S    init
    2     0 S    kthreadd
    3     2 I    kworker/0:0
    4     2 I<   kworker/0:0H
    5     2 I    kworker/u2:0
    6     2 I<   mm_percpu_wq
    7     2 S    ksoftirqd/0
    8     2 S    kdevtmpfs
    9     2 I<   netns
   10     2 S    oom_reaper
   11     2 I<   writeback
   12     2 I<   crypto
   13     2 I<   kblockd
   14     2 S    kswapd0
   15     2 I    kworker/0:1
   32     2 S    khvcd
   42     1 Ss   dhcpcd
   47     1 Ss   sh
   73     2 I    kworker/u2:1
  111    47 S+   a.out
  112   111 Z+   a.out <defunct>
  113   111 R+   ps
[root@localhost ~]#
```

b) Write a C/Java program to avoid zombie process by forking twice.

```
#include<stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
int main()
{
      pid_t pid1, pid2;
      if ((pid1=fork())< 0)
      {
            printf("Fork error");
      }
      else if( pid1==0)
      {
            printf("first child pid=%d\n", getpid());
            pid2=fork();
            if( pid2 > 0)
                exit(0);
            else if(pid2==0)
                        printf("second child pid = %d\n parent pid=%d\n",
getpid(), getppid());
            exit (0);
      }
}
```

Command for execution:

```
cc pgm_name.c
./a.out
```

```
[root@localhost ~]# cc 10b.c
[root@localhost ~]# ./a.out
[root@localhost ~]# first child pid=121
second child pid = 122
 parent pid=1
```

11. a) Write a C/C++ program to illustrate the race condition.

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
static void charatatime (char *);
int main (void)
{
pid_t pid;
if ((pid=fork( ))< 0)
{
      printf("fork error\n");
}
else if(pid==0)
{
      charatatime("Output from child\n");
}
else
{
      charatatime("Output from parent\n");
}
exit(0);
}
static void charatatime(char *str)
{
char *ptr;
int c;
setbuf(stdout,NULL); /* set unbuffered*/
for(ptr=str;(c=*ptr++)!=0;)
putc(c,stdout);
}
```

Command for execution:

```
cc pgm_name.c
./a.out
```

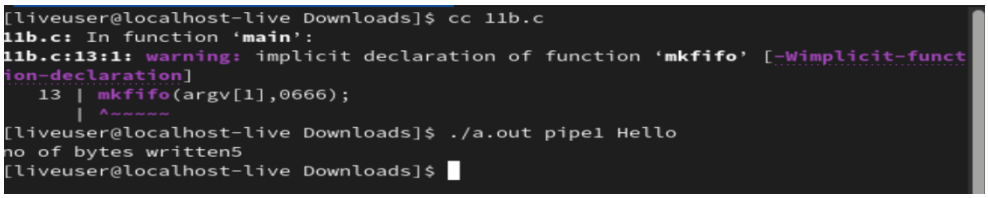
```
[root@localhost ~]# cc 11a.c
[root@localhost ~]# ./a.out
Output from pOutput from child
arent
[root@localhost ~]#
```

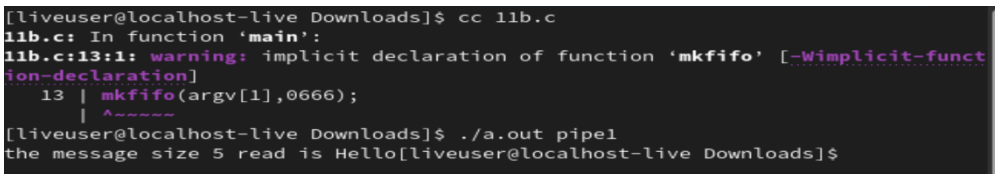
b) Write a C/C++ program which demonstrates inter-process communication between a reader process and a writer process. Use mkfifo, open, read, write and close APIs in your program.

```
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include<unistd.h>
#include<sys/types.h>
#include<string.h>
int main(int argc,char *argv[])
{
int fd,num1,num2;
char buf[100];
if(argc==3)
{
mkfifo(argv[1],0666);
fd=open(argv[1],O_WRONLY);
num1=write(fd,argv[2],strlen(argv[2]));
printf("no of bytes written%d\n",num1);
}
if(argc==2)
{
fd=open(argv[1],O_RDONLY);
num2=read(fd,buf,sizeof(buf));
buf[num2]='\0';
printf("the message size %d read is %s",num2,buf);
}
close(fd);
unlink(argv[1]);
return 0;
}
```

Command for execution:

```
Terminal -1
cc pgm_name.c
./a.out pipe1 Message

Terminal -2
cc pgm_name.c
./a.out pipe1
```





12. Design develop and run a multi-threaded program to generate and print Fibonacci series. One thread has to generate the numbers up to the specified limit and Another thread has to print them. Ensure proper synchronization.

```
#include<stdio.h>

#include<omp.h>

int main() {

   int n,a[100],i;
   omp_set_num_threads(2);
   printf("enter the no of terms of fibonacci series which have to be
generated\n");
   scanf("%d",&n);
   a[0]=0;
   a[1]=1;
   #pragma omp parallel
   {
        #pragma omp single
        for(i=2;i<n;i++)
        {
                a[i]=a[i-2]+a[i-1];
            printf("id of thread involved in the computation of fib no
%d is=%d\n",i+1,omp_get_thread_num());
        }
           #pragma omp barrier
           #pragma omp single
       {
                printf("the elements of fib series are\n");
                for(i=0;i<n;i++)
                printf("%d,id of the thread displaying this no is =
%d\n",a[i],omp_get_thread_num());
       }
   }
   return 0;
}
```

Command for execution:

```
cc -fopenmp 12.c
./a.out
```

```
[liveuser@localhost-live Downloads]$ cc -fopenmp 12.c
[liveuser@localhost-live Downloads]$ ./a.out
enter the no of terms of fibonacci series which have to be generated
8
id of thread involved in the computation of fib no 3 is=0
id of thread involved in the computation of fib no 4 is=0
id of thread involved in the computation of fib no 5 is=0
id of thread involved in the computation of fib no 6 is=0
id of thread involved in the computation of fib no 7 is=0
id of thread involved in the computation of fib no 8 is=0
the elements of fib series are
0,id of the thread displaying this no is =  1
1,id of the thread displaying this no is =  1
1,id of the thread displaying this no is =  1
2,id of the thread displaying this no is =  1
3,id of the thread displaying this no is =  1
5,id of the thread displaying this no is =  1
8,id of the thread displaying this no is =  1
13,id of the thread displaying this no is =  1
[liveuser@localhost-live Downloads]$
```