[Type the abstract of the document here. The abstract is typically a short summary of the contents of the document. Type the abstract of the document here. The abstract is typically a short summary of the contents of the document.]

[Type the document title]

[Type the document subtitle]

PROGRAM 1A:

PROGRAM TO COUNT THE NUMBER OF CHARACTERS, WORDS, SPACES AND LINES IN A GIVEN INPUT FILE.

LEX PROGRAM: 1a.lex

```
%{
#include<stdio.h>
int wc=0,cc=0,lc=0,bc=0;
%}
%%
[a-zA-Z]* {wc++; cc=cc+yyleng ;}
"\n" {lc++ ;}
[ ] {bc++ ;}
. {cc++ ;}
%%
int  main()
{
        FILE *fp;
        char file[50];
        printf("ENTER THE FILENAME:\n");
        scanf("%s",file);
        fp=fopen(file,"r");
        if(!fp)
        {
            fprintf(stderr,"FILE DOES NOT EXIST\n");
                exit(1);
         }
        yyin=fp;
        yylex();
        printf (" NO OF CHARACTERS=%d\n NO OF WORDS=%d\n
        NO OF LINES=%d\n NO OF  BLANKS=%d\n",cc,wc,lc,bc);
        return  0;
}
```

OUTPUT:

```
student@localhost:~/SS
File  Edit  View  Terminal  Tabs  Help
[student@localhost SS]$ vi 1aa.lex
[student@localhost SS]$ lex 1aa.lex
[student@localhost SS]$ cc lex.yy.c -ll
[student@localhost SS]$ cat file.txt
Jain Institute Of technology
[student@localhost SS]$ ./a.out
ENTER THE FILE NAME
file.txt
NO OF CHARACTERS=25
NO OF WORDS=4
NO OF LINES=1
NO OF BLANKS=3
[student@localhost SS]$ ./a.out
ENTER THE FILE NAME
file1.txt
NO OF CHARACTERS=35
NO OF WORDS=5
NO OF LINES=2
NO OF BLANKS=3
[student@localhost SS]$
```

TRY THIS CODE...

```
%%
[a-zA-z0-9]* {wc++;cc=cc+yyleng;}
“/n” {lc++;}
[ ] {cc++; bc++;}
. {cc++;}
%%
```

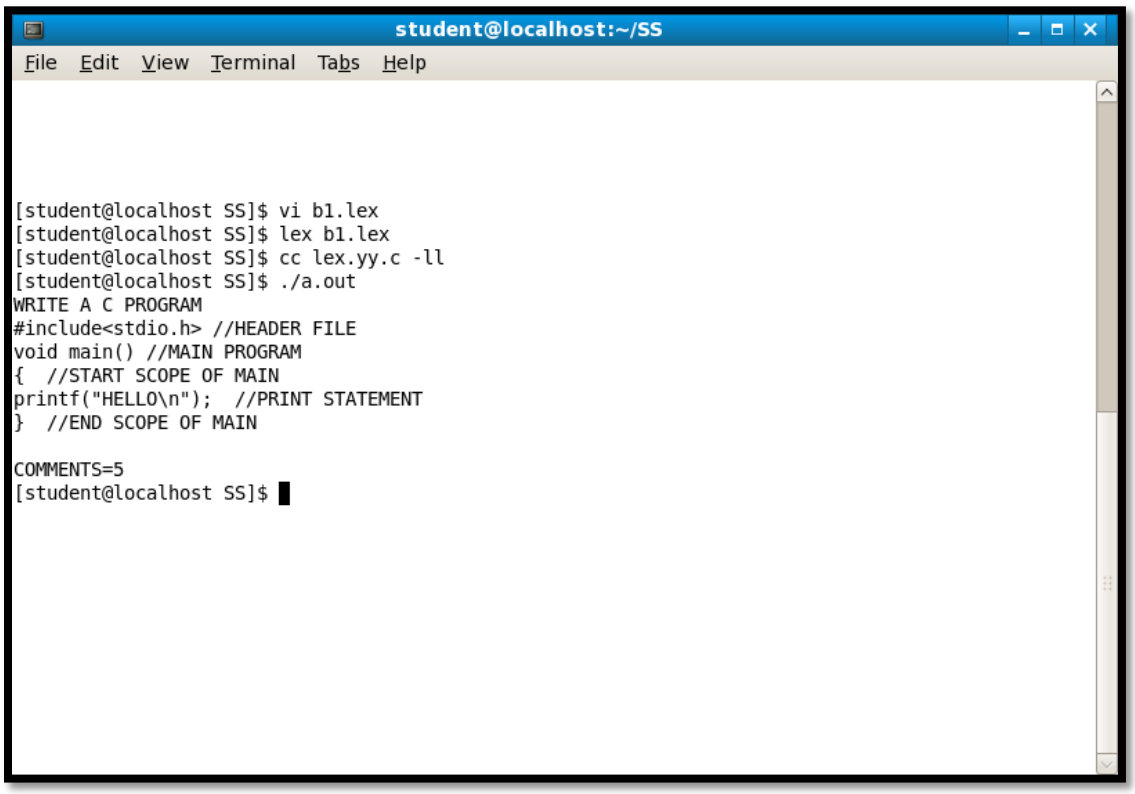
PROGRAM 1B:

PROGRAM TO COUNT THE NUMBERS OF COMMENT LINES IN A GIVEN C PROGRAM. ALSO ELIMINATE THEM AND COPY THE RESULTING PROGRAM INTO SEPARATE FILE.

LEX PROGRAM : 1b.lex

```
%{
#include<stdio.h>
int com=0;
%}
%%
"/*"[^"*/"]+"*/"  {com++;fprintf(yyout," ");}
"//"[^\n]+ {com++; fprintf(yyout, " ");}
%%

int main()
{
        printf("WRITE A C PROGRAM\n ");
        yyout=fopen("output","w");
        yylex();
        printf("COMMENT=%d\n",com);
        return 0;
}
```

OUTPUT:



➢ *Check it out…*

```
int main()
{
FILE *fp1,*fp2;
fp1=fopen("input.c","r");
fp2=fopen("output","w");
yyin=fp1;
yyout=fp2;
yylex();
printf("COMMENTS=%d",com);
}
```

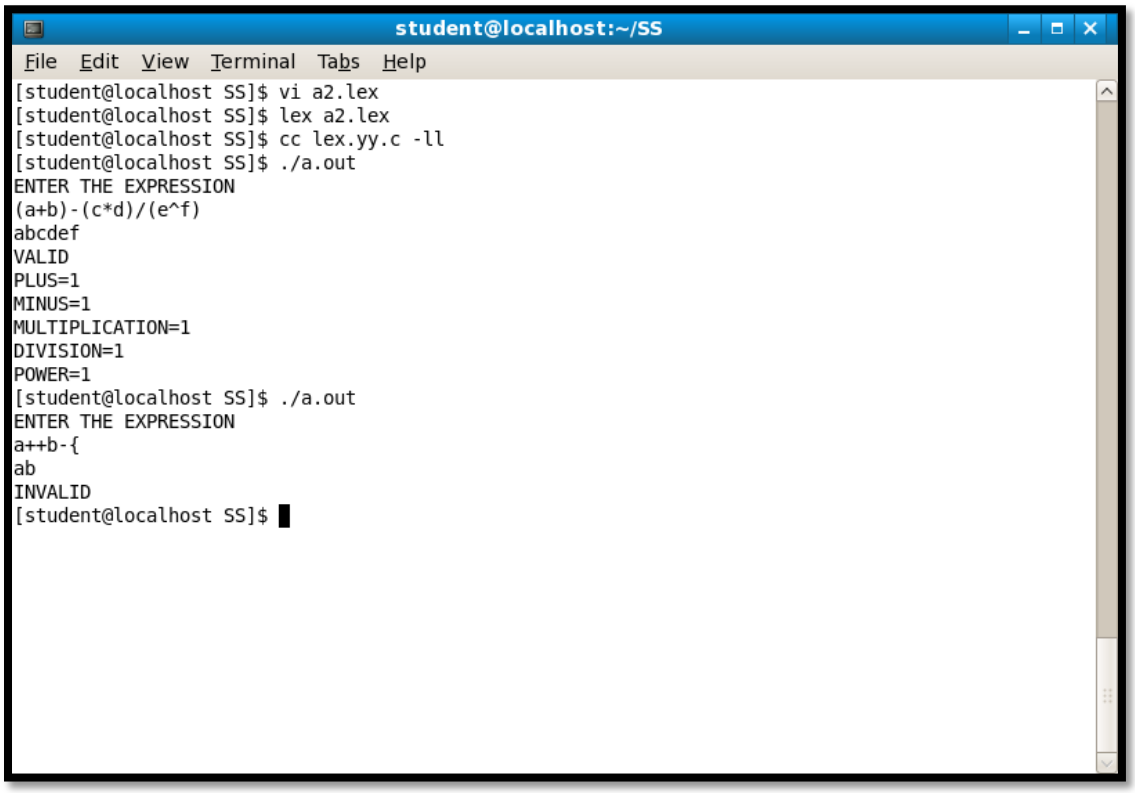
PROGRAM 2A:

PROGRAM TO RECOGIZNE A VALID ARITHMATIC EXPRESION AND TO RECOGNIZE IDENTIFIERS AND OPERATORS PRESENT.AND PRINT THEM SEPERATELY.

LEX PROGRAM: 2a.lex

```
%{
#include<stdio.h>
int invalid=0,p=0,m=0,n=0,ob=0,d=0,c=0;
%}
operand [a-zA-Z]+| [0-9]+
oprt "-"|[+*/^]
obr  "{"|"["|"("
cbr  "}"|"]"|")"
%%
{obr} {ob++ ;}
{cbr} {ob-- ;}
"+" {p++;}
"-" {n++;}
"*" {m++;}
"/" {d++;}
"^" {c++;}
{operand}{obr} {invalid=1;}
{oprt}{oprt}+|{oprt}{cbr} {invalid=1;}
{obr}{oprt}|{oprt}[\n] {invalid=1;}
{cbr}{operand}|{cbr} {invalid=1;}
{obr}|{cbr}{oprt}[\t\n] {invalid=1;}
%%
main()
{
        printf("ENTER THE EXPRESSION:\n");
        yylex();
        if(!invalid &&ob==0)
        {
         printf("VALID\n");
         printf("PLUS=%d\n MINUS=%d\n DIV=%d\n MUL=%d\n
        POW=%d\n",p,n,d,m,c);
        }
        else
        printf("INVALID\n");
}
```

OUTPUT:



Find it out??

- Take an expression from file as input

```
int main()
{
FILE *fp;
char file[10];
printf("ENTER THE FILE NAME\n");  /* expression is stored
                                        in file*/
scanf("%s",file);
fp=fopen(file,"r");
yylex();
//continue with the original programs
```

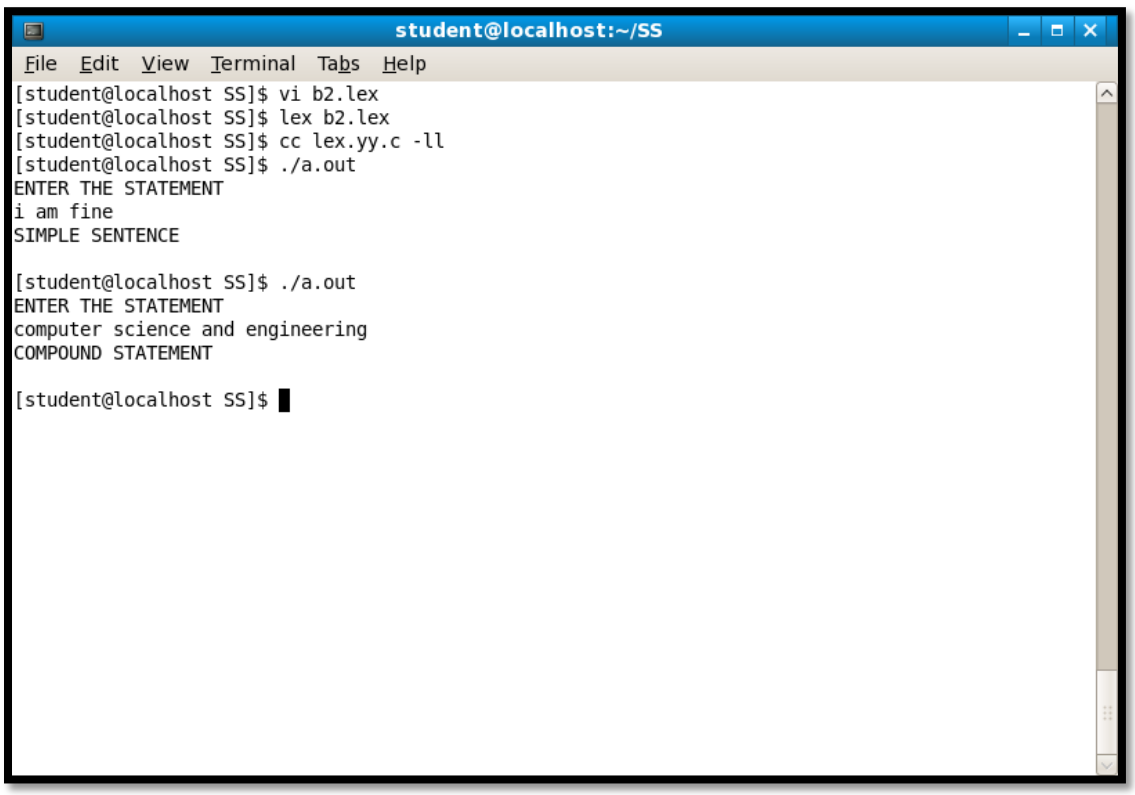
PROGRAM 2B:

PROGRAM TO RECOGNIZE WHETHER A GIVEN SENTENCE IS SIMPLE OR COMPOUND.

LEX PROGRAM: 2b.lex

```
%{
#include<stdio.h>
%}
%%
.+[ ]"and"+[ ].+ {printf("COMPOUND STATEMENT");}
"if"+[ ].+[ ]"then"+[ ].+ {printf("COMPOUND STATEMENT");}
.+[ ]"or"+[ ].+ {printf("COMPOUND STATEMENT");}
.+ {printf("SIMPLE STATEMENT");}
%%
main()
{
printf("ENTER THE SENTENCE\n");
yylex();
}
```

OUTPUT:



Try this Alternative:

.+[ ](“and|AND”)+[ ].+ {printf(“COMPOUND STATEMENT\n);}
(“if|IF”)+[ ].+[ ]+(“then|THEN”)+[ ].+ {printf(“COMPOUND STATEMENT\n”);
.+[ ](“or|OR”)+[ ].+ {printf(“COMPOUND STATEMENT\n”);}
.+ {printf(“SIMPLE STATEMENT\n”);

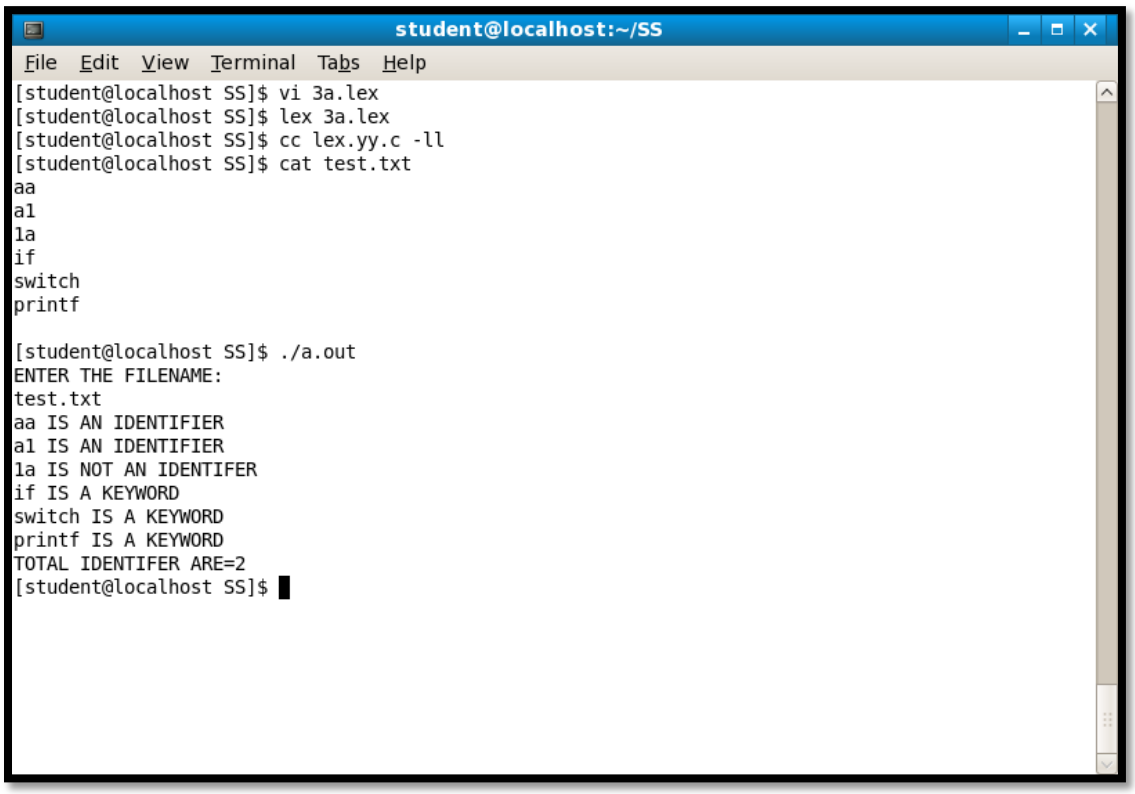
PROGRAM 3:

PROGRAM TO RECOGNIZE AND COUNT THE NUMBER OF IDENTIFIERS IN A GIVEN INPUT FILE.

LEX PROGRAM: 3.lex

```
%{
#include<stdio.h>
int count=0;
%}
op [+-*/]
letter [a-zA-Z]
digit [0-9]
id {letter}+({letter}|{digit})*
notid {digit}+{id}
%%
[\t\n]+ ;
("int") |
("float") |
("char") |
("case") |
("default") |
("if") |
(“else”) |
(“then”) |
(“while”) |
("for") |
("printf") |
("scanf")  {printf("%s IS A KEYWORD\n",yytext);}
{id} {printf("%s IS AN IDENTIFIER\n",yytext);count++;}
{notid} {printf("%s IS NOT AN IDENTIFIER\n",yytext);}
%%
 int main()
{
        FILE *fp;
        char file[10];
        printf("ENTER THE FILENAME:\n");
        scanf("%s",file);
        fp=fopen(file,"r");
        yyin=fp;
        yylex();
        printf("TOTAL IDENTIFIERS ARE=%d\n",count);
        return 0;
}
```

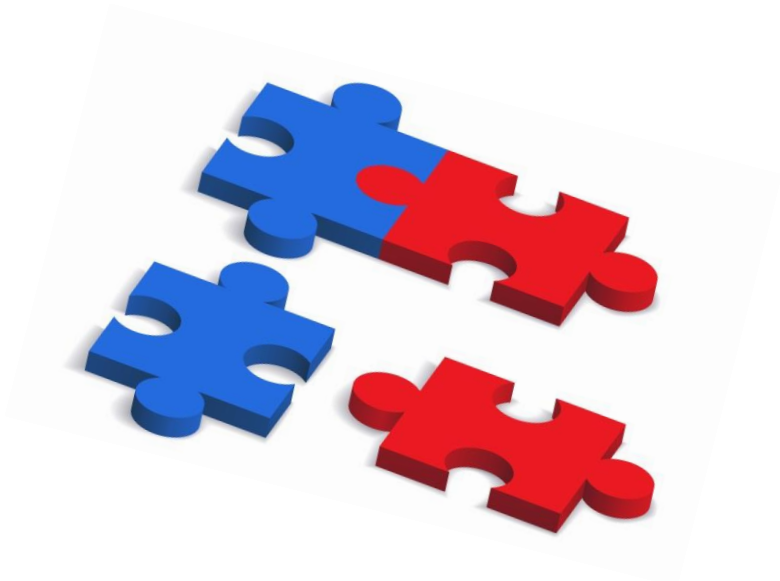
OUTPUT:

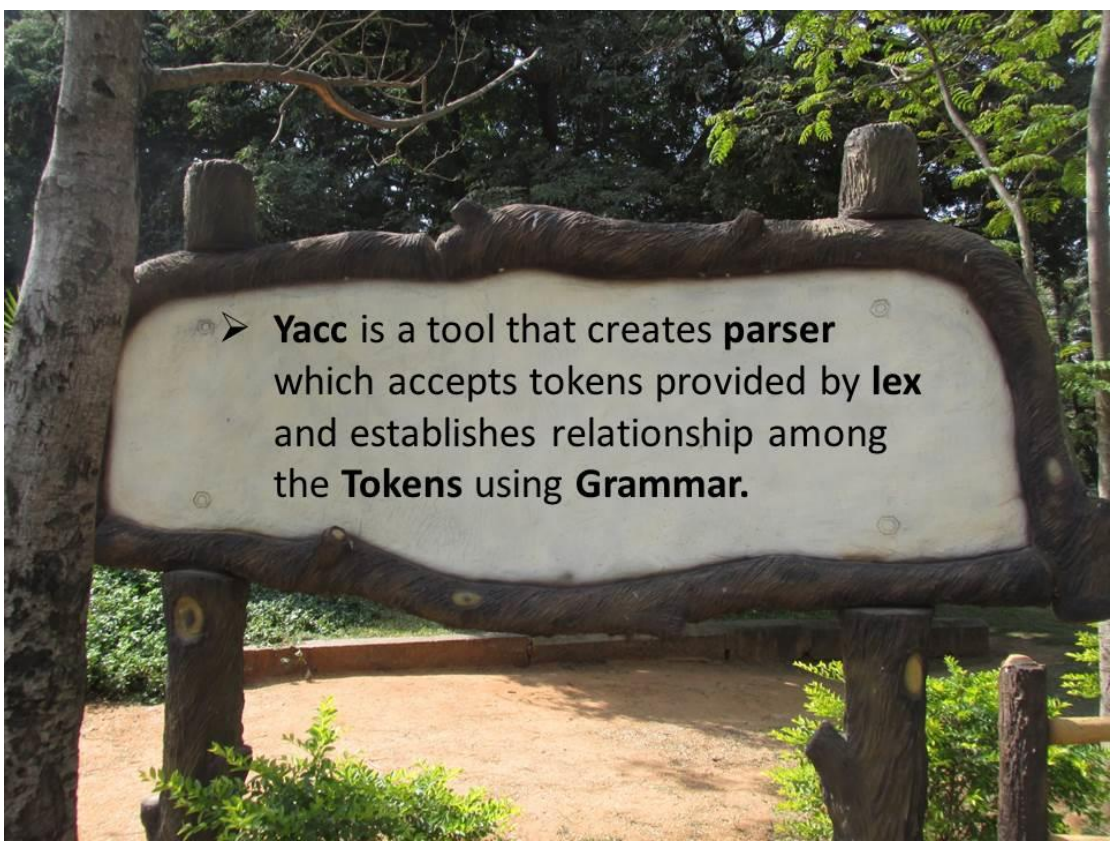


Check out:

symbol [@#$%^&*] //In definition section
notid {symbol}+{id} //in definition section
{notid} {printf("IS A SYMBOL");} //In rule section

Yacc programs





PROGARM 4A:

PROGRAM TO RECOGNIZE A VALID ARITHMETIC EXPRESSION THAT USES OPERATORS +,-,*, AND /.

LEX PROGARM: 4a.l

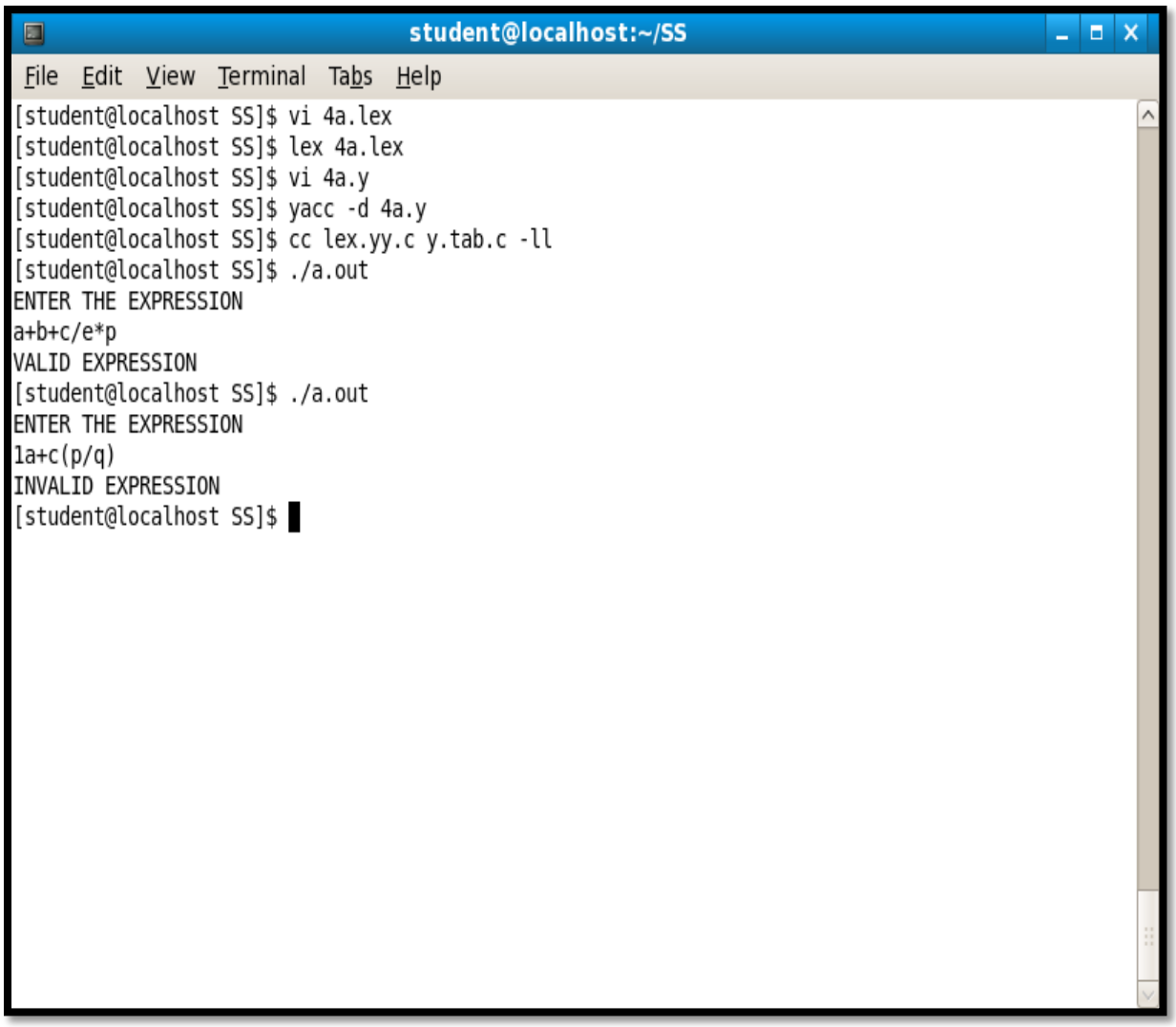
```
%{
#include<stdio.h>
#include"y.tab.h"
%}
%%
[0-9]+ {return(num);}
[a-zA-Z][a-zA-Z0-9]* {return(var);}
[+*/-] {return(yytext[0]);}
[\(\)] {return(yytext[0]);}
.|\n ;
%%
```

YACC PROGRAM: 4a.y

```
%{
#include<stdio.h>
int flag=0;
%}
%token  num var
%left '+"-'
%left '*"/'
%%
st:e {flag=1;}
e:e '+' e
 |e '-' e
 |e '*' e
 |e '/' e
 |'(' e ')'
 |num
 |var
;
%%
```

```
int main()
{
        printf("ENTER THE EXPRESSION\n");
        yyparse();
        if(flag)
        {
        printf("VALID EXPRESSION\n");
        }
}
yyerror()
{
        printf("INVALID EXPRESSION\n");
        exit(1);
}
```

OUTPUT:


student@localhost:~/SS
File Edit View Terminal Tabs Help
[student@localhost SS]$ vi 4a.lex
[student@localhost SS]$ lex 4a.lex
[student@localhost SS]$ vi 4a.y
[student@localhost SS]$ yacc -d 4a.y
[student@localhost SS]$ cc lex.yy.c y.tab.c -ll
[student@localhost SS]$ ./a.out
ENTER THE EXPRESSION
a+b+c/e*p
VALID EXPRESSION
[student@localhost SS]$ ./a.out
ENTER THE EXPRESSION
1a+c(p/q)
INVALID EXPRESSION
[student@localhost SS]$

PROGARM 4B:

PROGRAM TO RECOGNIZE A VALID VARIABLE, WHICH STARTS WITH A LETTER, FOLLOWED BY ANY NUMBER OF LETTERS OR DIGITS.

LEX PROGARM: 4b.l

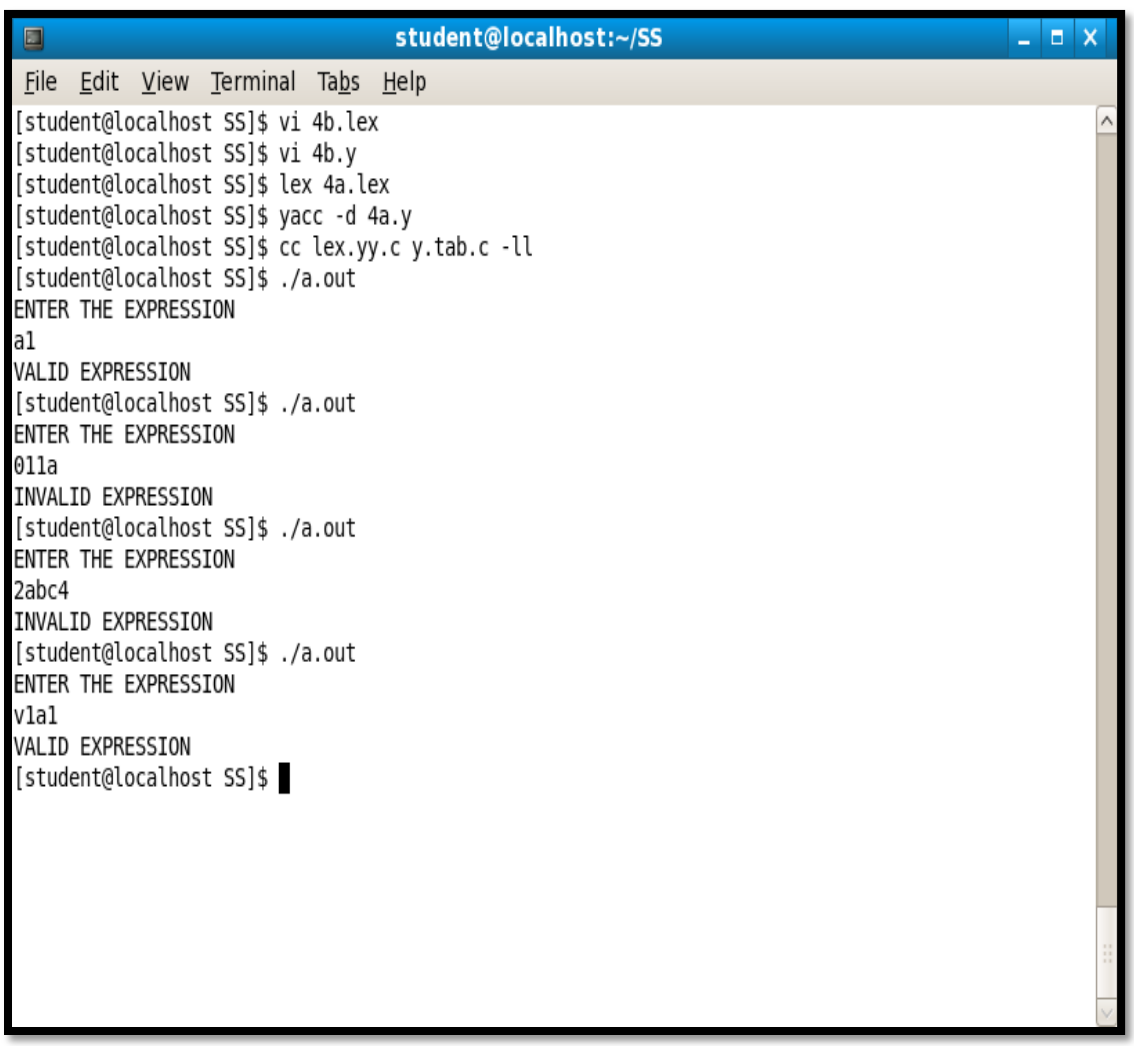
```
%{
#include"y.tab.h"
extern yylval;
%}
%%
[0-9]+ {yylval=atoi(yytext); return DIGIT;}
[a-zA-Z]+ {return LETTER;}
[\t] ;
\n return 0;
. {return yytext[0];}
%%
```

YACC PROGARM: 4b.y

```
%{
#include<stdio.h>
%}
%token LETTER DIGIT
%%
variable: LETTER|LETTER next
;
next: LETTER next
        |DIGIT next
        |LETTER
        |DIGIT
;
%%
main()
{
printf("ENTER THE VARIABLE\n”);
yyparse();
printf("THE STRING IS A VALID VARIABLE\n");
}
```

```
int yyerror(char *s)
{
printf("THIS IS NOT A VALID VARIABLE\n");
exit(0);
}
```

OUTPUT:


student@localhost:~/SS
File Edit View Terminal Tabs Help
[student@localhost SS]$ vi 4b.lex
[student@localhost SS]$ vi 4b.y
[student@localhost SS]$ lex 4a.lex
[student@localhost SS]$ yacc -d 4a.y
[student@localhost SS]$ cc lex.yy.c y.tab.c -ll
[student@localhost SS]$ ./a.out
ENTER THE EXPRESSION
a1
VALID EXPRESSION
[student@localhost SS]$ ./a.out
ENTER THE EXPRESSION
011a
INVALID EXPRESSION
[student@localhost SS]$ ./a.out
ENTER THE EXPRESSION
2abc4
INVALID EXPRESSION
[student@localhost SS]$ ./a.out
ENTER THE EXPRESSION
v1a1
VALID EXPRESSION
[student@localhost SS]$

PROGRAM 5A:

YACC PROGRAM TO EVALUATE AN ARITHMETIC EXPRESSION.

LEX PROGRAM: 5a.l

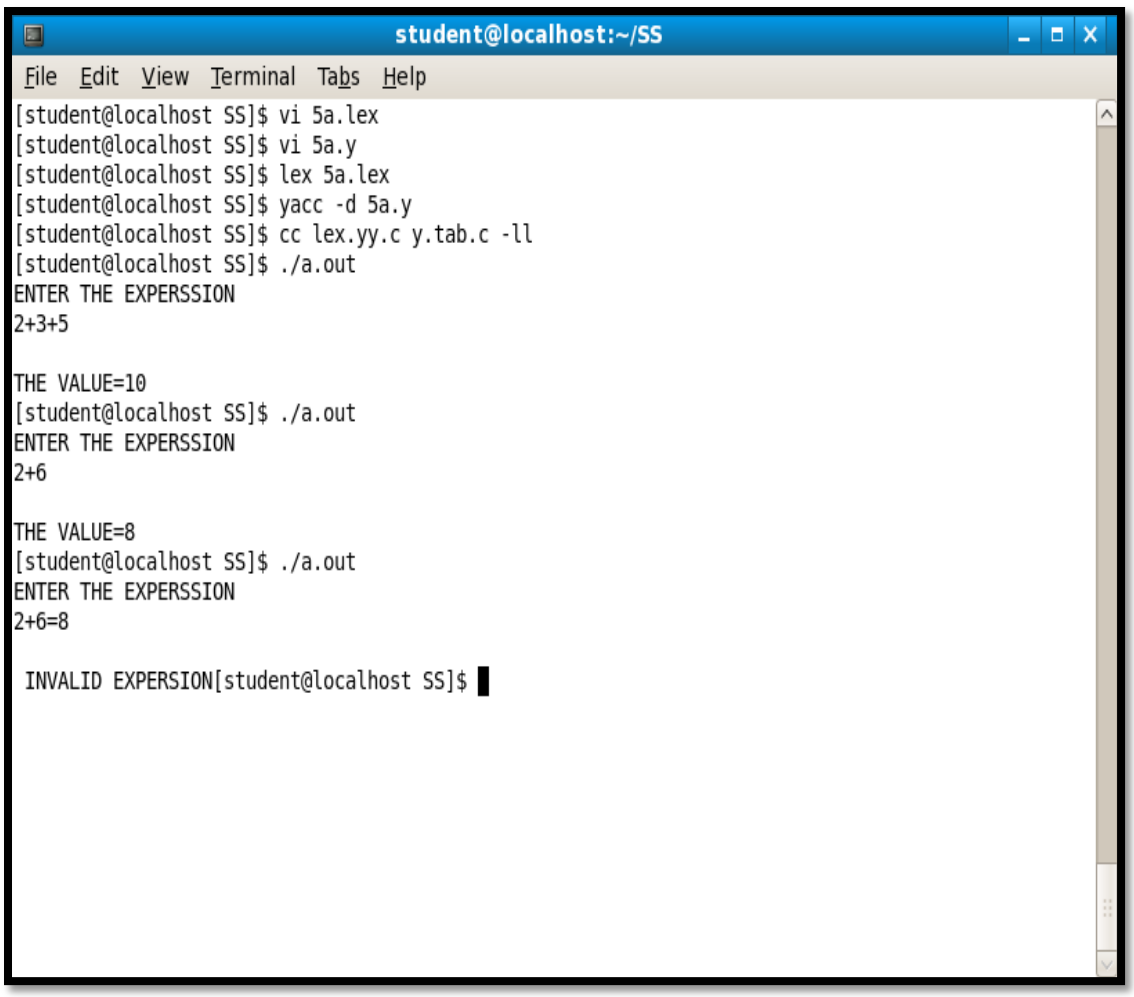
```
%{
#include<stdio.h>
#include"y.tab.h"
extern int yylval;
%}
%%
[0-9]+ {yylval=atoi(yytext);return(num);}
. {return(yytext[0]);}
\n {;}
%%
```

YACC PROGRAM:5a.y

```
%{
#include<stdio.h>
%}
%token  num
%left '+"-'
%left '*"/'
%%
start:expr {printf("\n THE VALUE=%d\n",$1);}
;
expr:'('expr')'{$$=$2;}
|
expr'+'expr {$$=$1+$3;}
|
expr'-'expr {$$=$1-$3;}
|
expr'*'expr {$$=$1*$3;}
|
expr'/'expr {$$=$1/$3;}
|
num {$$=$1;}
;
%%
```

```
main()
{
        printf("ENTER THE ARITHMETIC EXPRESSION\n");
        yyparse();
}
yyerror()
{
         printf("\n INVALID");
        exit(1);
}
```

OUTPUT:


student@localhost:~/SS
File Edit View Terminal Tabs Help
[student@localhost SS]$ vi 5a.lex
[student@localhost SS]$ vi 5a.y
[student@localhost SS]$ lex 5a.lex
[student@localhost SS]$ yacc -d 5a.y
[student@localhost SS]$ cc lex.yy.c y.tab.c -ll
[student@localhost SS]$ ./a.out
ENTER THE EXPERSSION
2+3+5
THE VALUE=10
[student@localhost SS]$ ./a.out
ENTER THE EXPERSSION
2+6
THE VALUE=8
[student@localhost SS]$ ./a.out
ENTER THE EXPERSSION
2+6=8
INVALID EXPERSION[student@localhost SS]$

PROGRAM5B:

YACC PROGRAM TO RECOGNIZE STRING 'aabb', 'ab', 'a AND 'aaab' USING THE GRAMMAR ($a^n b^n$, n>=0)

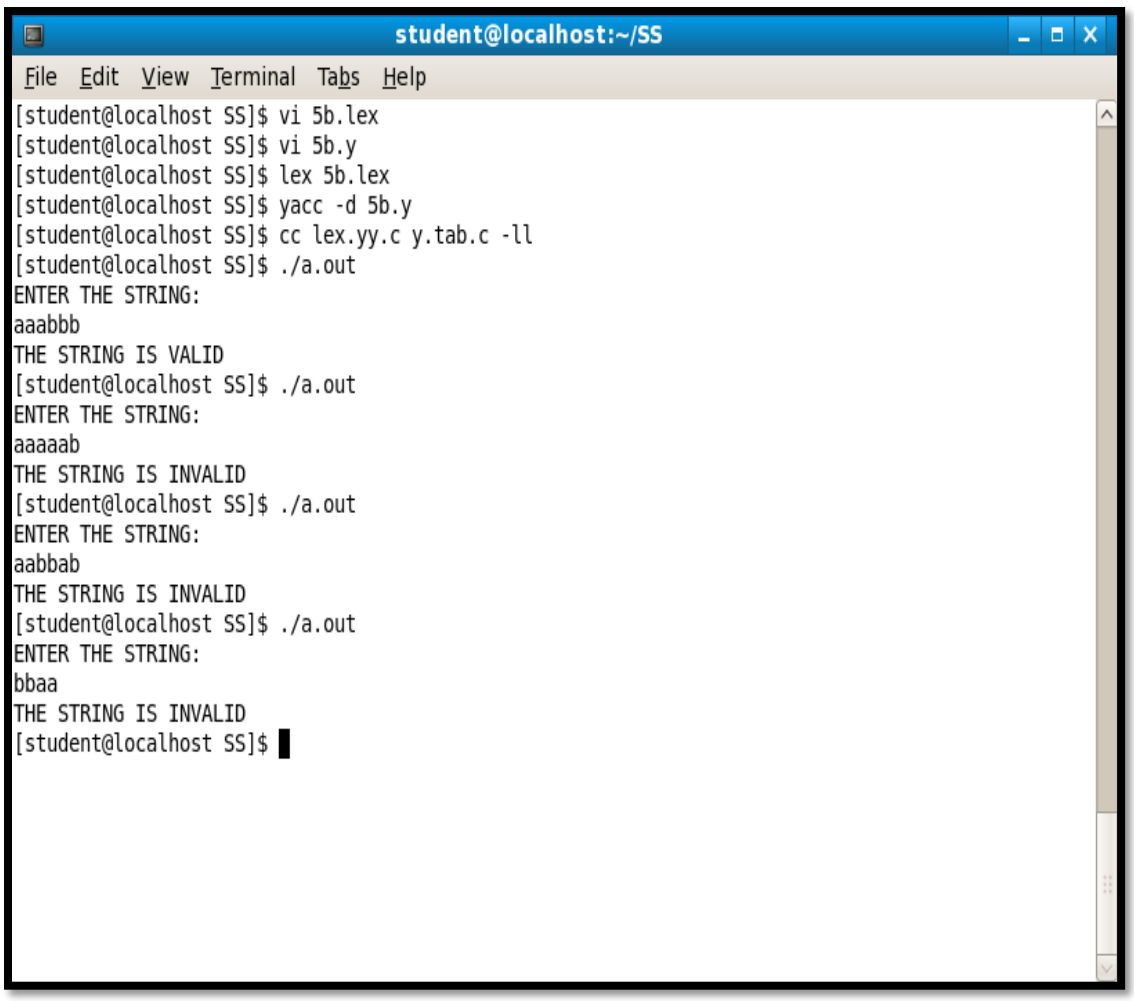
LEX PROGRAM:5b.l

```
%{
#include<stdio.h>
#include"y.tab.h"
%}
%%
a {return(A);}
b {return(B);}
[ c-z] {return(yytext[0]);}
.|\n {;}
%%
```

YACC PROGRAM: 5b.y

```
%{
#include<stdio.h>
int flag=0;
%}
%token A B
%%
t:s {flag=1;}
;
s:A s B
|
;
%%
int main()
{
        printf("ENTER THE STRING:\n");
        yyparse();
        if(flag)
                printf("THE STRING IS VALID\n");
}
yyerror()
{
        printf("INVAID STRING\n");
        exit(1);
}
```

OUTPUT:


student@localhost:~/SS
File Edit View Terminal Tabs Help
[student@localhost SS]$ vi 5b.lex
[student@localhost SS]$ vi 5b.y
[student@localhost SS]$ lex 5b.lex
[student@localhost SS]$ yacc -d 5b.y
[student@localhost SS]$ cc lex.yy.c y.tab.c -ll
[student@localhost SS]$ ./a.out
ENTER THE STRING:
aaabbb
THE STRING IS VALID
[student@localhost SS]$ ./a.out
ENTER THE STRING:
aaaaab
THE STRING IS INVALID
[student@localhost SS]$ ./a.out
ENTER THE STRING:
aabbab
THE STRING IS INVALID
[student@localhost SS]$ ./a.out
ENTER THE STRING:
bbaa
THE STRING IS INVALID
[student@localhost SS]$

PROGRAM 6:

YACC PROGRAM TO RECOGNIZE THE GRAMMAR $a^n b$, n>=10.

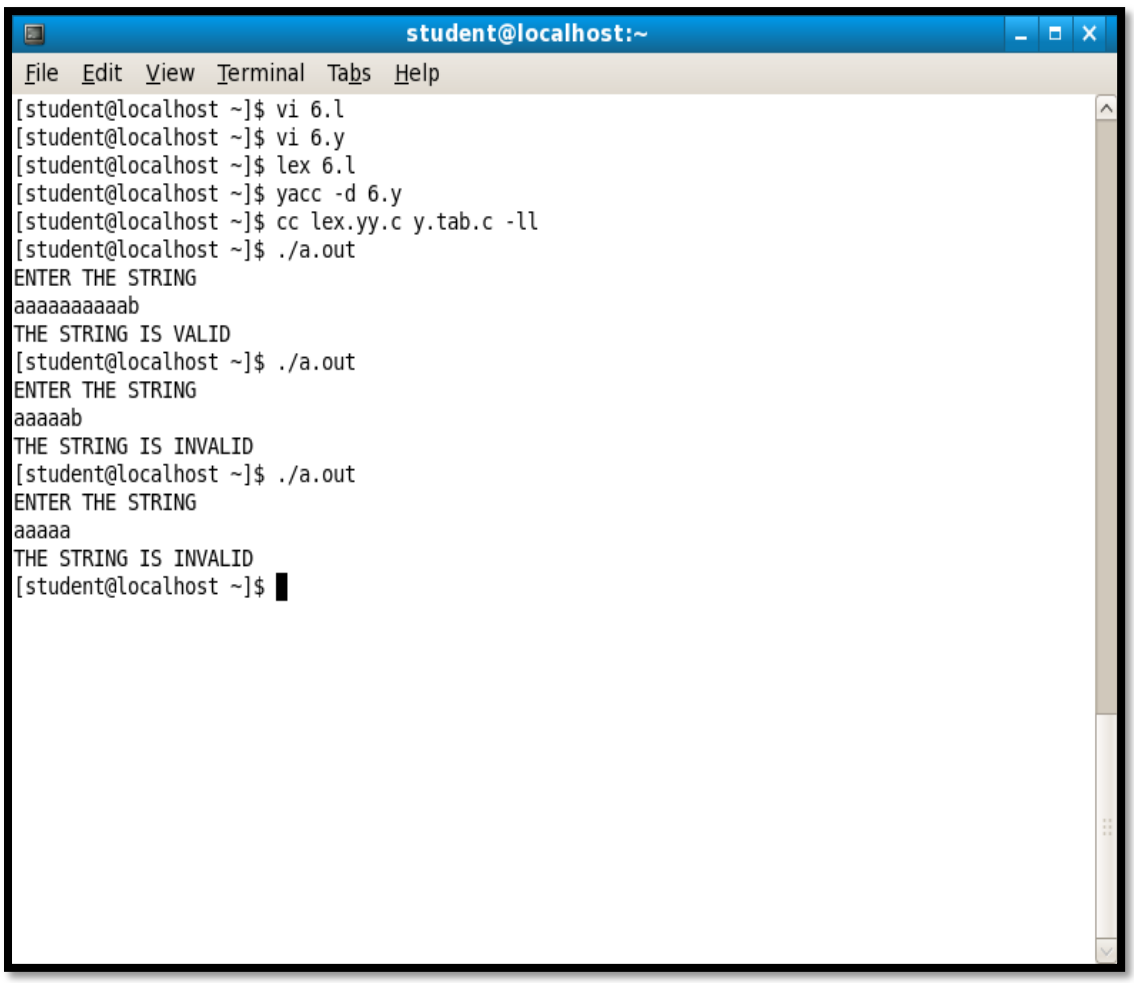
LEX PROGRAM: 6.l

```
%{
#include<stdio.h>
#include"y.tab.h"
%}
%%
aaaaaaaaaa+ {return(A);}
b {return(B);}
[c-z ] {return(yytext[0]);}
.|\n {;}
%%
```

YACC PROGRAM: 6.y

```
%{
#include<stdio.h>
int flag=0;
%}
%token A B
%%
t:s {flag=1;}
;
s:A B
;
%%
main()
{
        printf("\n ENTER THE STRING:\n\n");
        yyparse();
        if(flag)
                printf("\n THE STRING IS VALID\n\n");
}
yyerror()
{
        printf("INVALID\n");
        exit(1);
}
```

OUTPUT:


student@localhost:~
File Edit View Terminal Tabs Help
[student@localhost ~]$ vi 6.l
[student@localhost ~]$ vi 6.y
[student@localhost ~]$ lex 6.l
[student@localhost ~]$ yacc -d 6.y
[student@localhost ~]$ cc lex.yy.c y.tab.c -ll
[student@localhost ~]$ ./a.out
ENTER THE STRING
aaaaaaaaaab
THE STRING IS VALID
[student@localhost ~]$ ./a.out
ENTER THE STRING
aaaaab
THE STRING IS INVALID
[student@localhost ~]$ ./a.out
ENTER THE STRING
aaaaa
THE STRING IS INVALID
[student@localhost ~]$

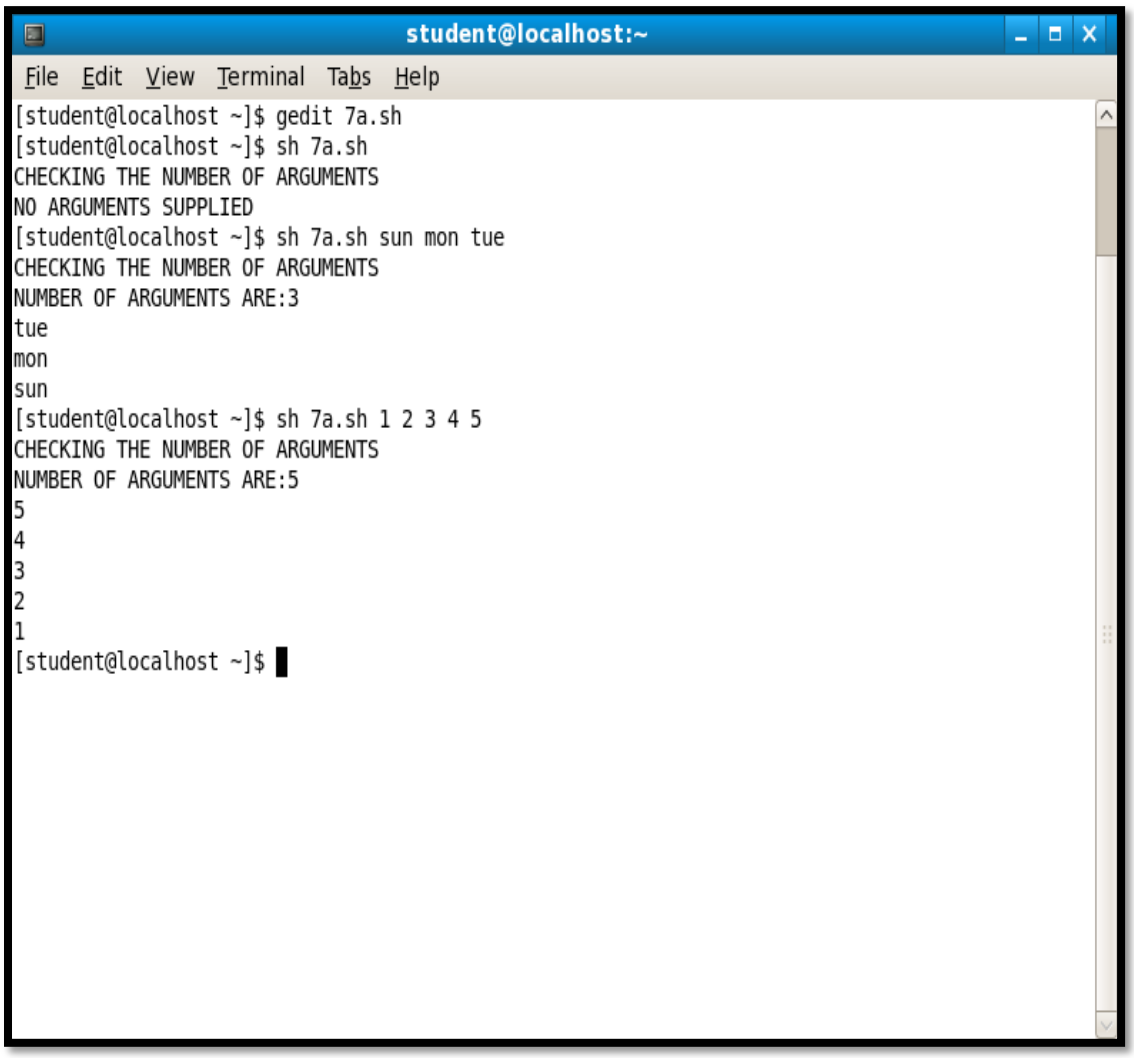
PROGRAM 7A:

NON RECURSIVE SHELL SCRIPT THAT ACCEPTS ANY NUMBER OF ARGUMENTS AND PRINTS THEM IN THE REVERSE ORDER.

SHELL PROGRAM: 7a.sh

```
echo "CHECKING THE NUMBER OF ARGUMENTS"
if [  $#  -eq  0  ]
then
echo "NO ARGUMENTS SUPPLIED"
exit
fi
echo "NO OF ARGUMENTS ARE:$#"
len=$#
while [  $len  –gt  0  ]
do
eval echo \$$len
len=  ` expr  $len  –  1 `
done
```

OUTPUT:


student@localhost:~
File Edit View Terminal Tabs Help
[student@localhost ~]$ gedit 7a.sh
[student@localhost ~]$ sh 7a.sh
CHECKING THE NUMBER OF ARGUMENTS
NO ARGUMENTS SUPPLIED
[student@localhost ~]$ sh 7a.sh sun mon tue
CHECKING THE NUMBER OF ARGUMENTS
NUMBER OF ARGUMENTS ARE:3
tue
mon
sun
[student@localhost ~]$ sh 7a.sh 1 2 3 4 5
CHECKING THE NUMBER OF ARGUMENTS
NUMBER OF ARGUMENTS ARE:5
5
4
3
2
1
[student@localhost ~]$

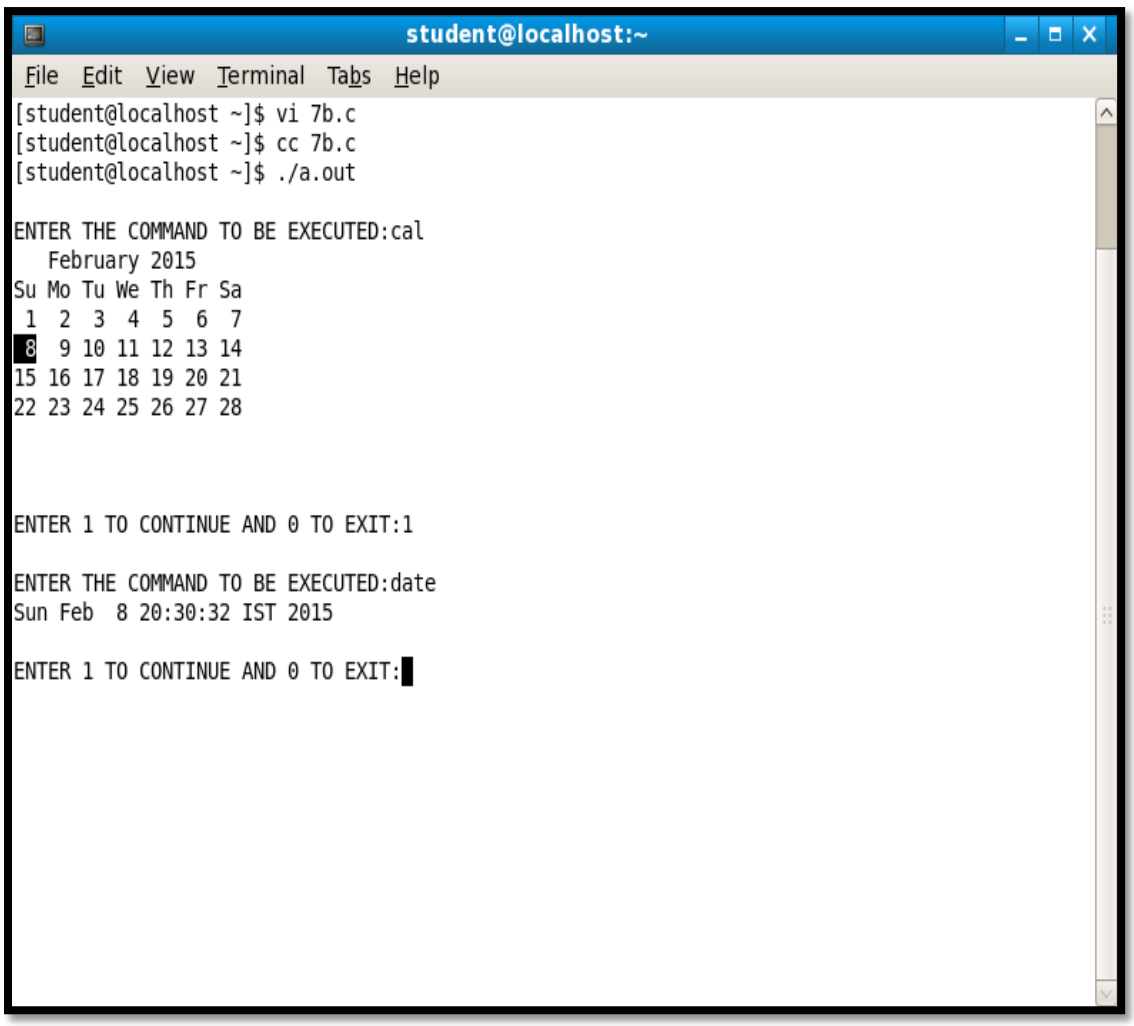
PROGARM 7B:

PROGRAM THAT CREATES A CHILD PROCESS TO READ COMMANDS FROM THE STANDARD INPUTS AND EXECUTE THEM.YOU CAN ASSUME THAT NO ARGUMENTS WILL BE PASSED TO THE COMMANDS TO BE EXECUTED.

PROGRAM: 7b.c

```
#include<stdio.h>
#include<sys/types.h>
#include<stdlib.h>
int main()
{
        char cmd[20];
        pid_t pid;
        int ch;
        pid=fork();
        if(pid==0)
         {
         do
         {
          printf("\n ENTER THE COMMAND TO BE EXECUTED:");
          scanf("%s",cmd);
          system(cmd);
          printf("\n ENTER 1 TO CONTINUE AND 0 TO EXIT:");
          scanf("%d",&ch);
         }while(ch!=0);
         }
wait();
}
```

OUTPUT:


student@localhost:~
File Edit View Terminal Tabs Help
[student@localhost ~]$ vi 7b.c
[student@localhost ~]$ cc 7b.c
[student@localhost ~]$ ./a.out
ENTER THE COMMAND TO BE EXECUTED:cal
February 2015
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
ENTER 1 TO CONTINUE AND 0 TO EXIT:1
ENTER THE COMMAND TO BE EXECUTED:date
Sun Feb 8 20:30:32 IST 2015
ENTER 1 TO CONTINUE AND 0 TO EXIT:

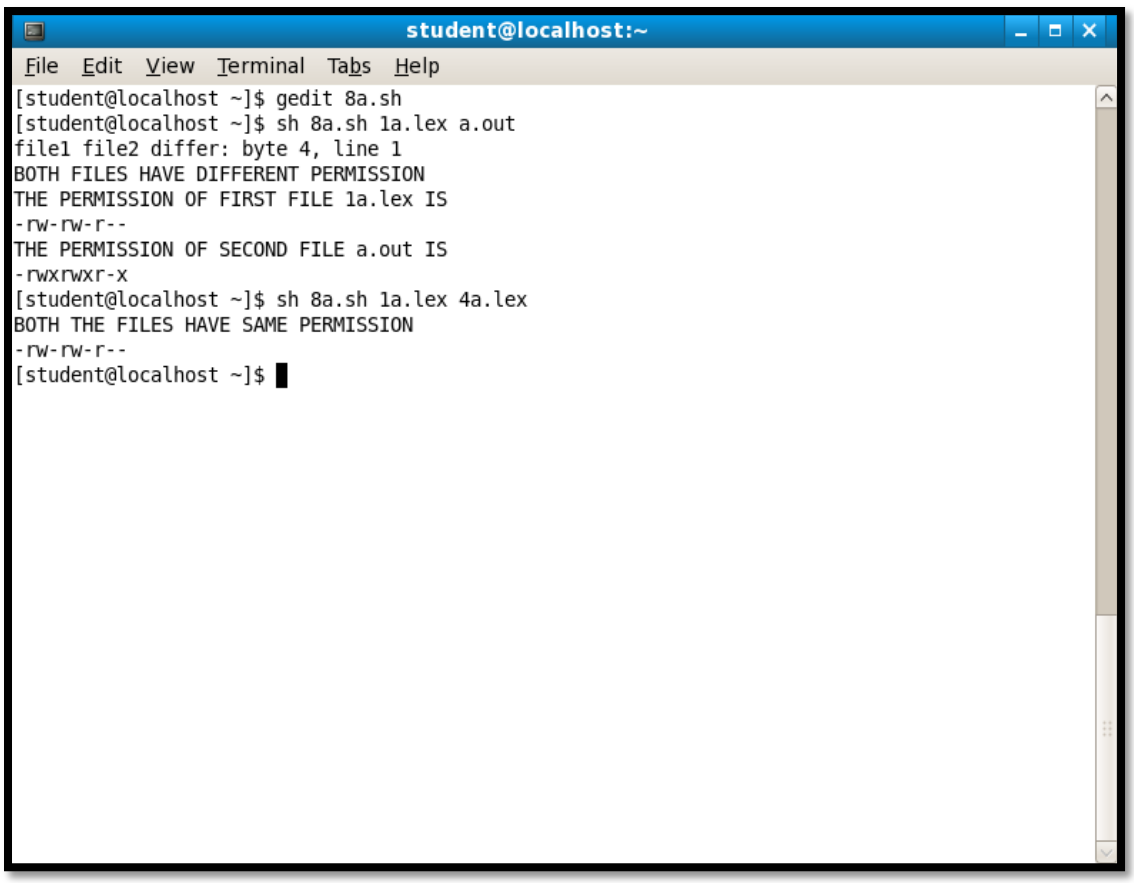
PROGRAM 8A:

SHELL SCRIPT THAT ACCEPTS TWO FILE NAMES AS ARGUMENTS, CHECKS IF THE PERMISSIONS FOR THESE FILES ARE IDENTICAL AND IF SAME OUTPUT COMMON PERMISSION OTHERWISE OUTPUT EACH FILE NAME FOLLOWED BY ITS PERMISSIONS.

SHELL SCRIPT: 8a.sh

```
ls -l $1 | cut -d " " -f1 > file1
ls -l $2 | cut -d " " -f1 > file2
if cmp file1 file2
then
echo "BOTH THE FILES HAVE SAME PERMISSION"
cat file1
else
echo "BOTH THE FILES DIFFERENT PERMISSION"
echo "THE PERMISSION OF FIRST FILE $1 IS"
cat file1
echo "THE PERMISSION OF SECOND FILE $2 IS"
cat file2
fi
```

OUTPUT:


student@localhost:~
File Edit View Terminal Tabs Help
[student@localhost ~]$ gedit 8a.sh
[student@localhost ~]$ sh 8a.sh 1a.lex a.out
file1 file2 differ: byte 4, line 1
BOTH FILES HAVE DIFFERENT PERMISSION
THE PERMISSION OF FIRST FILE 1a.lex IS
-rw-rw-r--
THE PERMISSION OF SECOND FILE a.out IS
-rwxrwxr-x
[student@localhost ~]$ sh 8a.sh 1a.lex 4a.lex
BOTH THE FILES HAVE SAME PERMISSION
-rw-rw-r--
[student@localhost ~]$

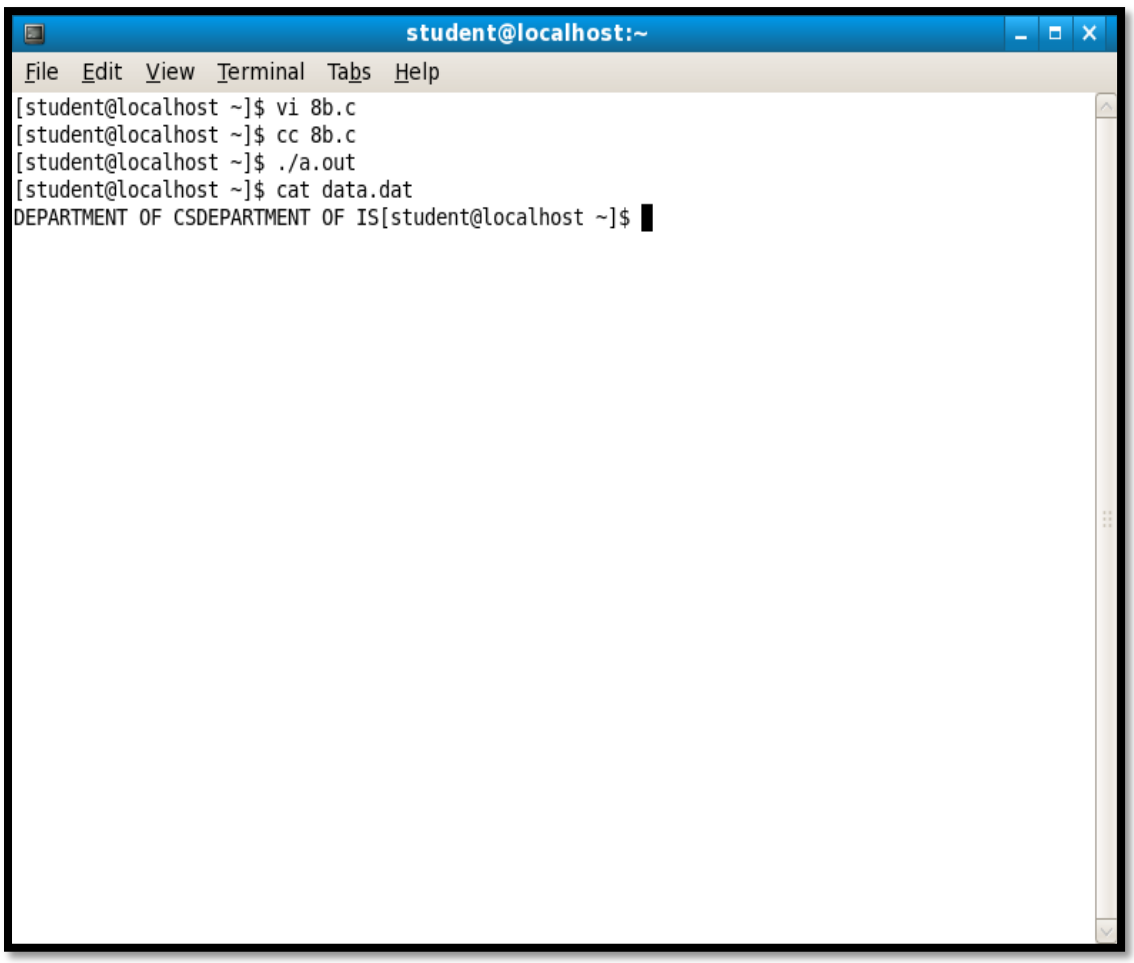
PROGRAM 8B:

C PROGRAM TO CREATE A FILE WITH 16-BYTES OF ARBITRARY DATA FROM THE BEGINNING AND ANOTHER 16-BYTES OF ARBITRARY DATA FROM AN OFFSET OF 48. DISPLAY THE FILE CONTENTS TO DEMONSTRATE HOW THE HOLE IN IS HANDLED.

C PROGRAM : 8b.c

```
#include<stdio.h>
int main()
{
        FILE *fd;
        char buf1[]="Department of CS";   /*THERE SHOULD BE A
                                            TOTAL 16 CHARACTERS*/
        char buf2[]="Department of IS";   // IN BOTH BUF1 AND BUF2
        fd=fopen("data.dat","w");
        if(fd<0)
        {
                printf("\n ERROR IN CREATING FILE");
                exit(0);
        }
        fprintf(fd,"%s",buf1);
        fseek(fd, 48, 0);
        fprintf(fd,"%s",buf2);
        exit(0);
}
```

OUTPUT:


student@localhost:~
File Edit View Terminal Tabs Help
[student@localhost ~]$ vi 8b.c
[student@localhost ~]$ cc 8b.c
[student@localhost ~]$ ./a.out
[student@localhost ~]$ cat data.dat
DEPARTMENT OF CSDEPARTMENT OF IS[student@localhost ~]$

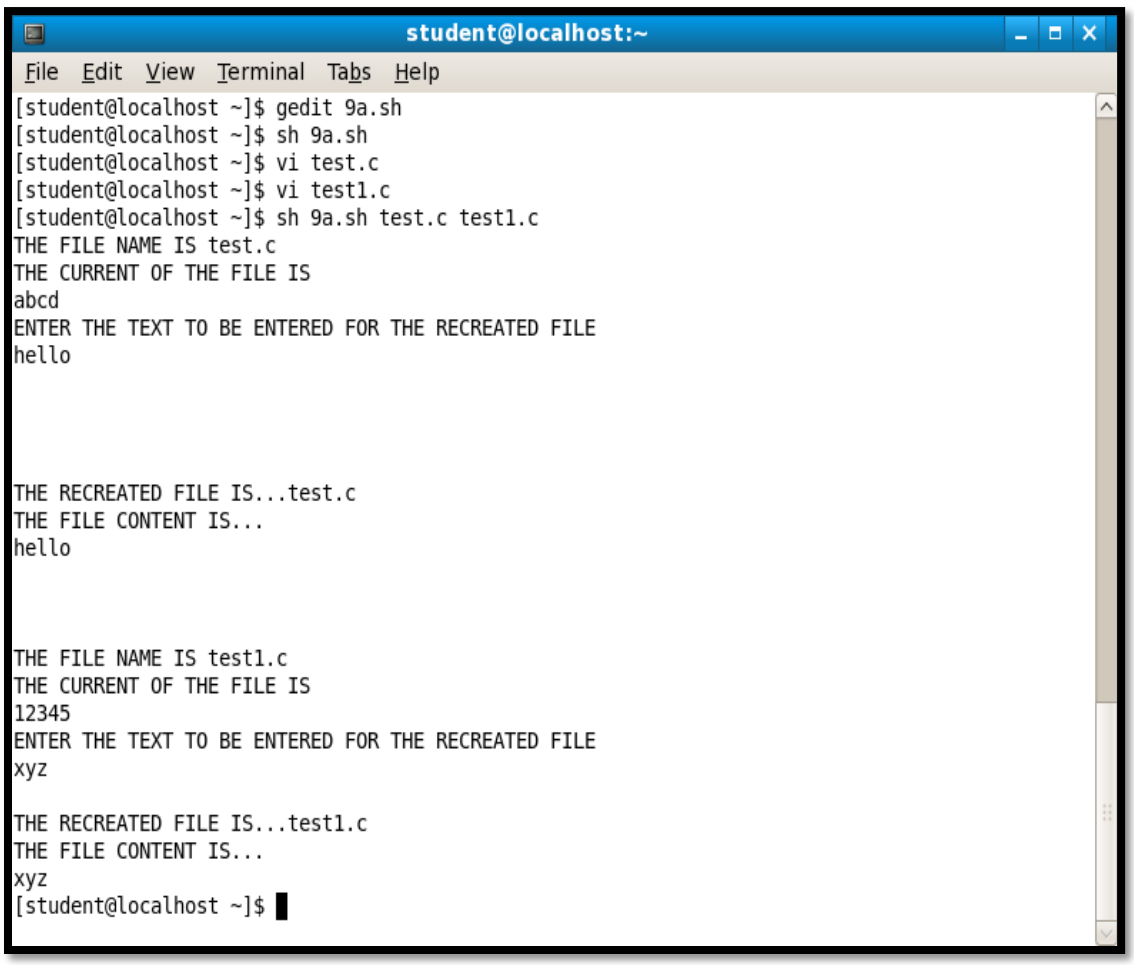
PROGRAM 9A:

SHELL SCRIPT THAT ACCEPTS FILE NAMES SPECIFIED AS ARGUMENTS AND CREATES A SHELL SCRIPTS THAT CONTAINS THIS FILE AS WELL AS THE CODE TO RECREATE THESE FILES. THUS IF THE SCRIPT GENERATED BY YOUR SCRIPT IS EXECUTED, IT WOULD RECREATE THE ORIGINAL FILES.

SHELL PROGRAM: 9a.sh

```
for i in $*
do
echo "THE FILENAME IS $i"
echo "THE CURRENT CONTENT OF THE FILE IS"
cat $i FILE
echo "ENTER THE TEXT TO BE ENTERED FOR THE RECREATED"
echo `cat > $i`
echo "THE RECREATED FILE IS.....$i"
echo "THE FILE CONTENT IS....."
cat $i
done
```

OUTPUT:


student@localhost:~
File Edit View Terminal Tabs Help
[student@localhost ~]$ gedit 9a.sh
[student@localhost ~]$ sh 9a.sh
[student@localhost ~]$ vi test.c
[student@localhost ~]$ vi test1.c
[student@localhost ~]$ sh 9a.sh test.c test1.c
THE FILE NAME IS test.c
THE CURRENT OF THE FILE IS
abcd
ENTER THE TEXT TO BE ENTERED FOR THE RECREATED FILE
hello
THE RECREATED FILE IS...test.c
THE FILE CONTENT IS...
hello
THE FILE NAME IS test1.c
THE CURRENT OF THE FILE IS
12345
ENTER THE TEXT TO BE ENTERED FOR THE RECREATED FILE
xyz
THE RECREATED FILE IS...test1.c
THE FILE CONTENT IS...
xyz
[student@localhost ~]$

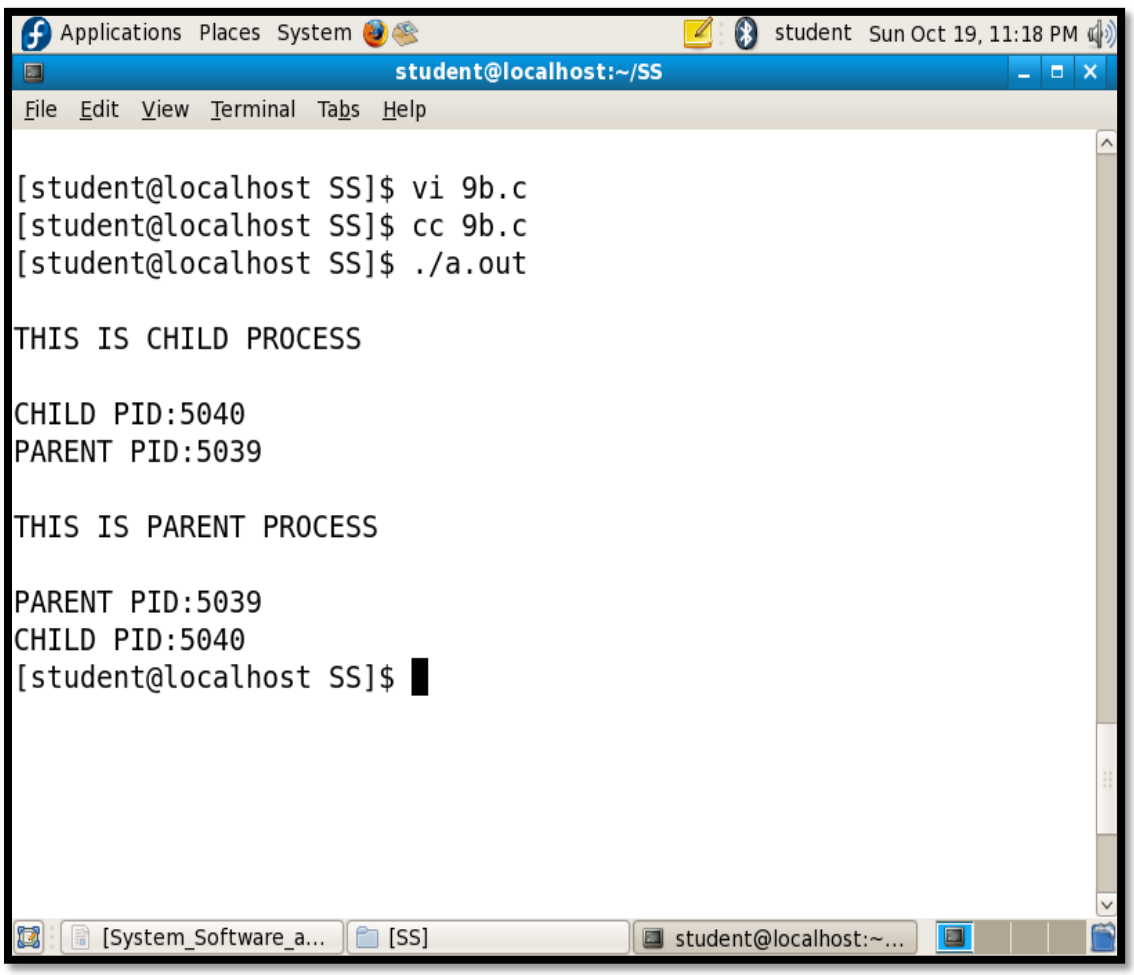
PROGRAM 9B:

C PROGRAM TO DO THE FOLLOWING:USING FORK() CREATE A CHILD PROCESS. THE CHILD PROCESS PRINTS ITS OWN PROCESS ID AND ID OF ITS PARENT AND THEN EXITS. THE PARENT PROCESS WAITS FOR ITS CHILD TO FINISH BY EXECUTING THE WAIT() AND PRINTS ITS OWN PROCESS ID AND ID OF ITS CHILD PROCESS AND EXITS.

C PROGRAM: 9b.c

```
#include<sys/types.h>
#include<stdio.h>
int main()
{
        Pid _t pid;
        if((pid=fork())<0)
        printf("FORK ERROR \n");
        if(pid==0)
        {
                printf("\nTHIS IS CHILD PROCESS\n");
                printf("\nCHILD PID:%d\n",getpid());
                printf("\nPARENT PID:%d\n",getppid());
                exit(0);
        }
        else
        {
                wait();
                printf("\nTHIS IS PARENT PROCESS");
                printf("\nPARENT PID:%d",getpid());
                printf("\nCHILD PID:%d",pid);
                exit(0);
        }
}
```

OUTPUT:


Applications Places System
student Sun Oct 19, 11:18 PM
student@localhost:~/SS
File Edit View Terminal Tabs Help
[student@localhost SS]$ vi 9b.c
[student@localhost SS]$ cc 9b.c
[student@localhost SS]$ ./a.out
THIS IS CHILD PROCESS
CHILD PID:5040
PARENT PID:5039
THIS IS PARENT PROCESS
PARENT PID:5039
CHILD PID:5040
[student@localhost SS]$
[System_Software_a...
[SS]
student@localhost:~...

PROGRAM 10:

DESIGN, DEVELOP AND EXECUTE A PROGRAM IN C/C++ TO SIMULATE THE WORKING OF SHORTEST REMAINING TIME AND ROUND ROBIN SCHEDULING ALGORITHMS. EXPERIMENT WITH DIFFERENT QUANTUM SIZES FOR THE ROUND ROBIN ALGORITHM. IN ALL CASES, DETERMINE THE AVERAGE TURN AROUND TIME. INPUT CAN BE READ FROM KEYBOARD OR FROM A FILE.

```
#include<stdio.h>
#include<stdlib.h>
struct proc
{
        int id;
        int arrival;
        int burst;
        int rem;
        int wait;
        int finish;
        int turnaround;
        float ratio;
}
process[10];
struct proc temp;
int no;

int chkprocess(int);
int nextprocess();
void roundrobin(int,int,int[],int[]);
void srtf(int);

main()
{
        int n,tq,choice;
        int bt[10],st[10],i,j,k;
        for(; ;)
        {
                printf("ENTER THE CHOICE \n");
                printf("1.ROUND ROBIN\n2.SRT\n3.EXIT\n");
```

```
                scanf("%d",&choice);
                switch(choice)
                {
case 1:
printf("ROUND ROUBIN SCHEDULING ALGORITHM\n");
printf("ENTER NUMBER OF PROCESSES:\n");
scanf("%d",&n);
printf("ENTER BURST TIME FOR SEQUENCES:");
for(i=0;i<n;i++)
{
        scanf("%d",&bt[i]);
        st[i]=bt[i];
}
printf("ENTER TIME QUANTUM:");
scanf("%d",&tq);
roundrobin(n,tq,st,bt);
break;

case 2:
printf("\n\n---SHORTEST REMAINING TIME NEXT---\n\n");
printf("\n\n ENTER THE NUMBER OF PROCESSES:\n\n");
scanf("%d",&n);
srtf(n);
break;

case 3:
exit(0);
  }
 }
}

void roundrobin(int n, int tq, int st[], int bt[])
{
        int time=0;
        int tat[10],wt[10],i,count=0,swt=0,stat=0,temp1,sq=0,j,k;
        float awt=0.0,atat=0.0;
        while(1)
        {
                for(i=0,count=0;i<n;i++)
                {
                        temp1=tq;
                        if(st[i]==0)
                        {
```

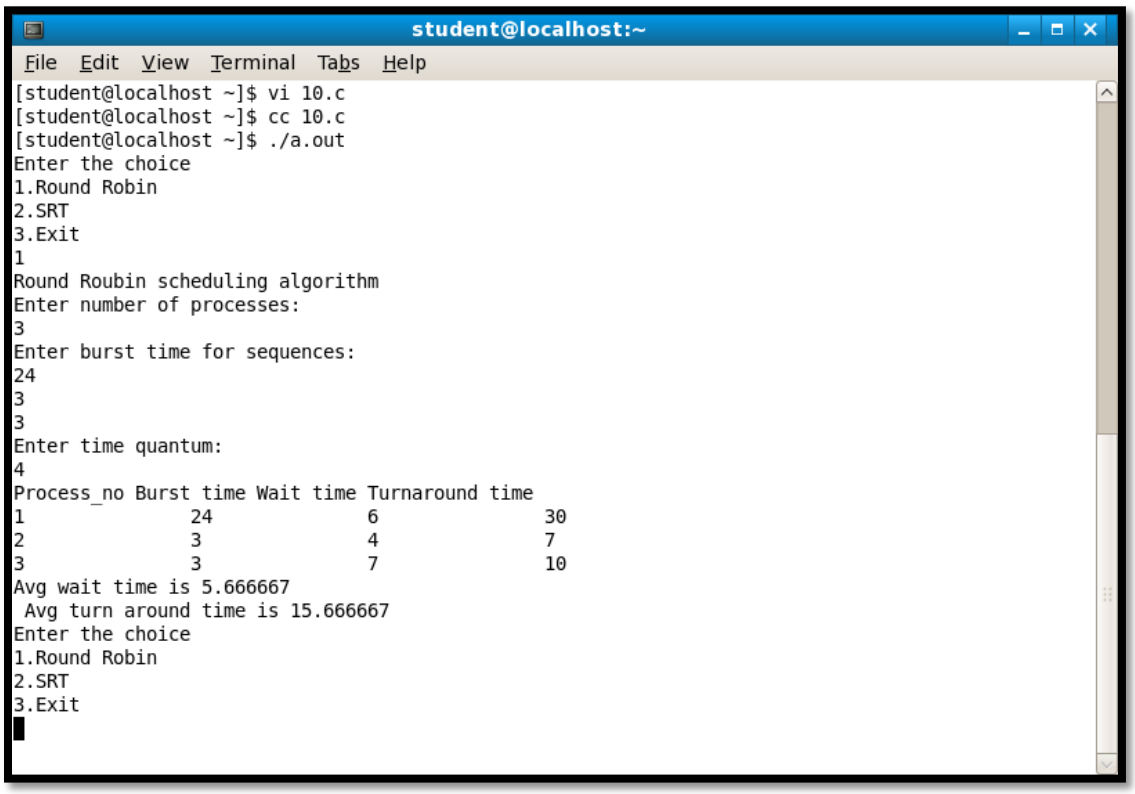
```
                        count++;
                        continue;
                }
                if(st[i]>tq)
                st[i]=st[i]-tq;
                else
                if(st[i]>=0)
                {
                        temp1=st[i];
                        st[i]=0;
                }
                sq=sq+temp1;
                tat[i]=sq;
        }
        if(n==count)
        break;
}
for(i=0;i<n;i++)
{
        wt[i]=tat[i]-bt[i];
        swt=swt+wt[i];
        stat=stat+tat[i];
}
awt=(float)swt/n;
atat=(float)stat/n;
printf("PROCESS_NO  BURST TIME  WAIT TIME  TURNAROUND
TIME\n");
for(i=0;i<n;i++)
printf("%d\t\t%d\t\t%d\t\t%d\n",i+1,bt[i],wt[i],tat[i]);
printf("AVG WAIT TIME IS %f\N AVG TURN AROUND TIME IS
%f\n",awt,atat);
}

int chkprocess(int s)
{
        int i;
        for(i=1;i<=s;i++)
        {
                if(process[i].rem!=0)
                return 1;
        }
        return 0;
}
```

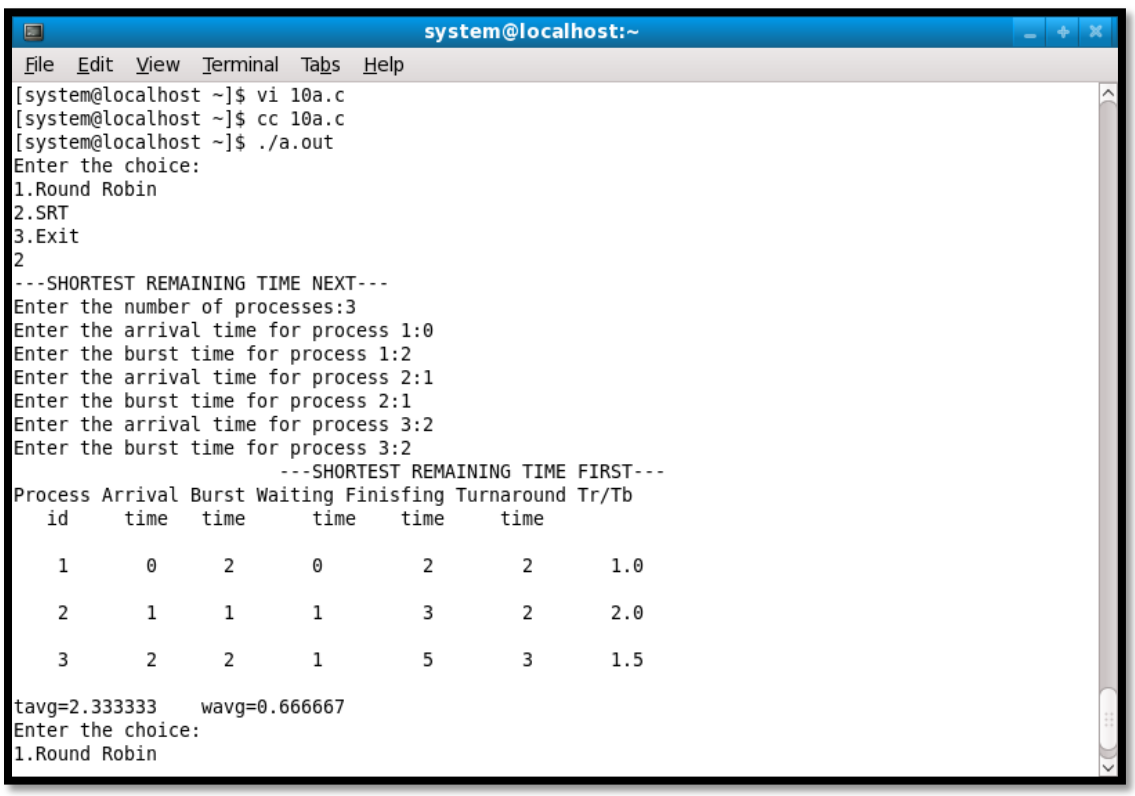
```
int nextprocess()
{
	int min,l,i;
	min=32000;
	for(i=1;i<=no;i++)
	{
		if(process[i].rem!=0 && process[i].rem<min)
		{
			min=process[i].rem;
			l=i;
		}
	}
	return l;
}

void srtf(int n)
{
	int i,j,k,time=0;
	float tavg,wavg;
	for(i=1;i<=n;i++)
	{
	process[i].id=i;
	printf("\nENTER THE ARRIVAL TIME FOR PROCESS %d:",i);
	scanf("%d",&(process[i].arrival));
	printf("\nENTER THE BURST TIME FOR PROCESS %d:",i);
	scanf("%d",&(process[i].burst));
	process[i].rem=process[i].burst;
	}
	for(i=1;i<=n;i++)
	{
		for(j=i+1;j<=n;j++)
		{
			if(process[i].arrival>process[j].arrival)
			{
				temp=process[i];
				process[i]=process[j];
				process[j]=temp;
			}
		}
	}
	no=0;
	j=1;
```

```
while(chkprocess(n)==1)
{
	if(process[no+1].arrival==time)
	{
		no++;
		if(process[j].rem==0)
		process[j].finish=time;
		j=nextprocess();
	}
	if(process[j].rem!=0)
	{
		process[j].rem--;
		for(i=1;i<=no;i++)
		{
			if(i!=j && process[i].rem!=0)
			process[i].wait++;
		}
	}
	else
	{
		process[j].finish=time;
		j=nextprocess();
		time--;
		k=j;
	}
	time++;
}
process[k].finish=time;
printf("\n\n\t\t\t---SHORTEST REMAINING TIME FIRST---");
printf("\n\nPROCESS ARRIVAL BURST WAITING FINISFING
TURNAROUND Tr/Tb\n");
printf("%5s%9s%7s%10s%8s%9s\n\n","id","time","time","time","time",
"time");
for(i=1;i<=n;i++)
{
	process[i].turnaround=process[i].wait+process[i].burst;
	process[i].ratio=(float)process[i].turnaround/(float)process[i].burst;
	printf("%5d%8d%7d%8d%10d%9d%10.1f",process[i].id,process[i
	].arrival,process[i].burst,process[i].wait,process[i].finish,process[i].
	turnaround,process[i].ratio);
	tavg=tavg+process[i].turnaround;
	wavg=wavg+process[i].wait;
```

```
        printf("\n\n");
}
tavg=tavg/n;
wavg=wavg/n;
printf("TAVG=%f\t WAVG=%f\n",tavg,wavg);
}
```

OUTPUT :


student@localhost:~
File Edit View Terminal Tabs Help
[student@localhost ~]$ vi 10.c
[student@localhost ~]$ cc 10.c
[student@localhost ~]$ ./a.out
Enter the choice
1.Round Robin
2.SRT
3.Exit
1
Round Roubin scheduling algorithm
Enter number of processes:
3
Enter burst time for sequences:
24
3
3
Enter time quantum:
4
Process_no Burst time Wait time Turnaround time
1 24 6 30
2 3 4 7
3 3 7 10
Avg wait time is 5.666667
Avg turn around time is 15.666667
Enter the choice
1.Round Robin
2.SRT
3.Exit

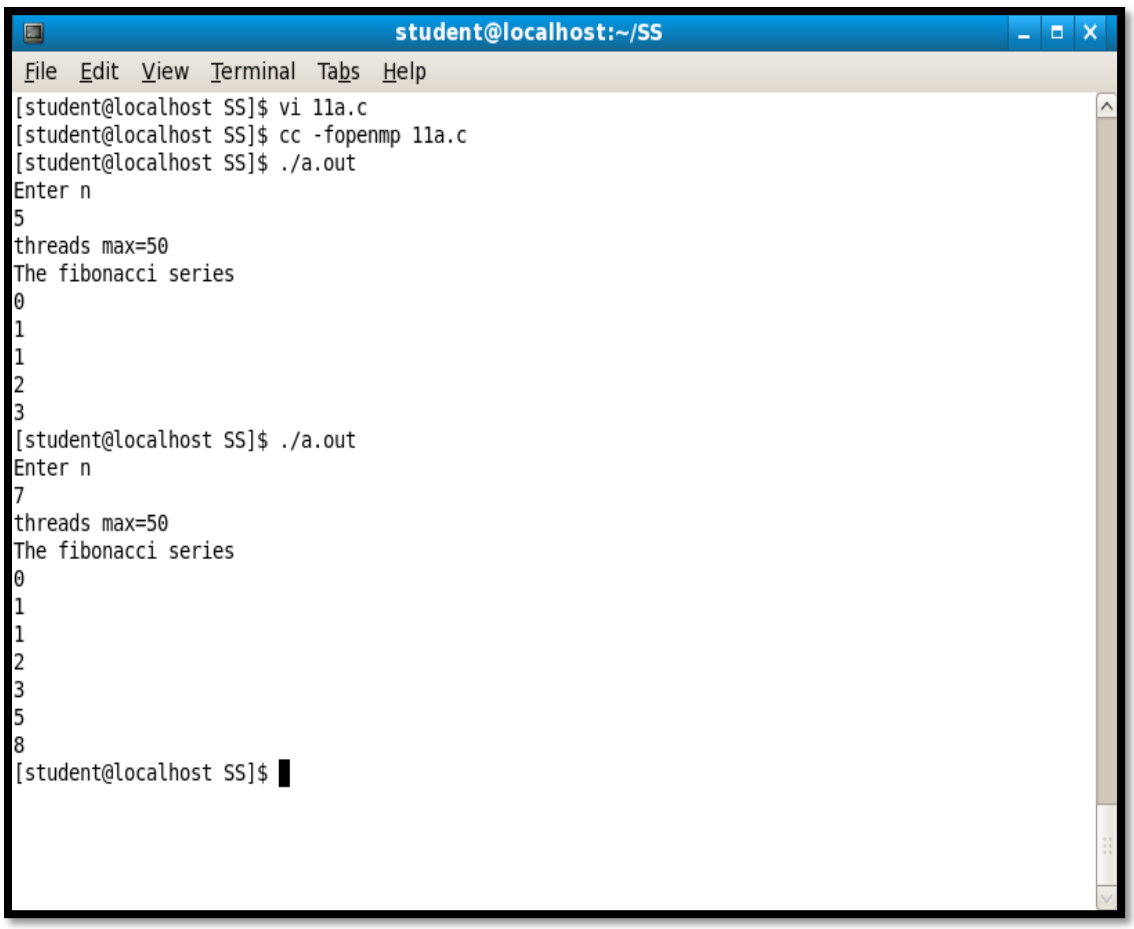

system@localhost:~
File Edit View Terminal Tabs Help
[system@localhost ~]$ vi 10a.c
[system@localhost ~]$ cc 10a.c
[system@localhost ~]$ ./a.out
Enter the choice:
1.Round Robin
2.SRT
3.Exit
2
---SHORTEST REMAINING TIME NEXT---
Enter the number of processes:3
Enter the arrival time for process 1:0
Enter the burst time for process 1:2
Enter the arrival time for process 2:1
Enter the burst time for process 2:1
Enter the arrival time for process 3:2
Enter the burst time for process 3:2
---SHORTEST REMAINING TIME FIRST---
Process Arrival Burst Waiting Finisfing Turnaround Tr/Tb
id time time time time time
1 0 2 0 2 2 1.0
2 1 1 1 3 2 2.0
3 2 2 1 5 3 1.5
tavg=2.333333 wavg=0.666667
Enter the choice:
1.Round Robin

PROGRAM 11:

USING OPEN MP 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>
#define N 100
#define NUM_THREADS 50

int main()
{
        int i,n,a[100];
        printf("ENTER A FIBONACCI NUMBER:\n");
        scanf("%d",&n);
        a[0]=0,a[1]=1;
        omp_set_num_threads(NUM_THREADS);
        printf("THREADS MAX=%d\n",omp_get_max_threads());
        #pragma omp parallel
        for(i=2;i<n;i++)
        {
                a[i]=a[i-1]+a[i-2];
        }
        printf("THE FIBONACCI SERIES\n");
        for(i=0;i<n;i++)
        {
                printf("%d\n",a[i]);
        }
                return 0;
}
```

OUTPUT :


student@localhost:~/SS
File Edit View Terminal Tabs Help
[student@localhost SS]$ vi 11a.c
[student@localhost SS]$ cc -fopenmp 11a.c
[student@localhost SS]$ ./a.out
Enter n
5
threads max=50
The fibonacci series
0
1
1
2
3
[student@localhost SS]$ ./a.out
Enter n
7
threads max=50
The fibonacci series
0
1
1
2
3
5
8
[student@localhost SS]$

PROGRAM 12:

DESIGN, DEVELOP AND RUN A PROGRAM TO IMPLEMENT THE BANKER'S ALGORITHM. DEMONSTRATE ITS WORKING WITH DIFFERENT DATA VALUES.

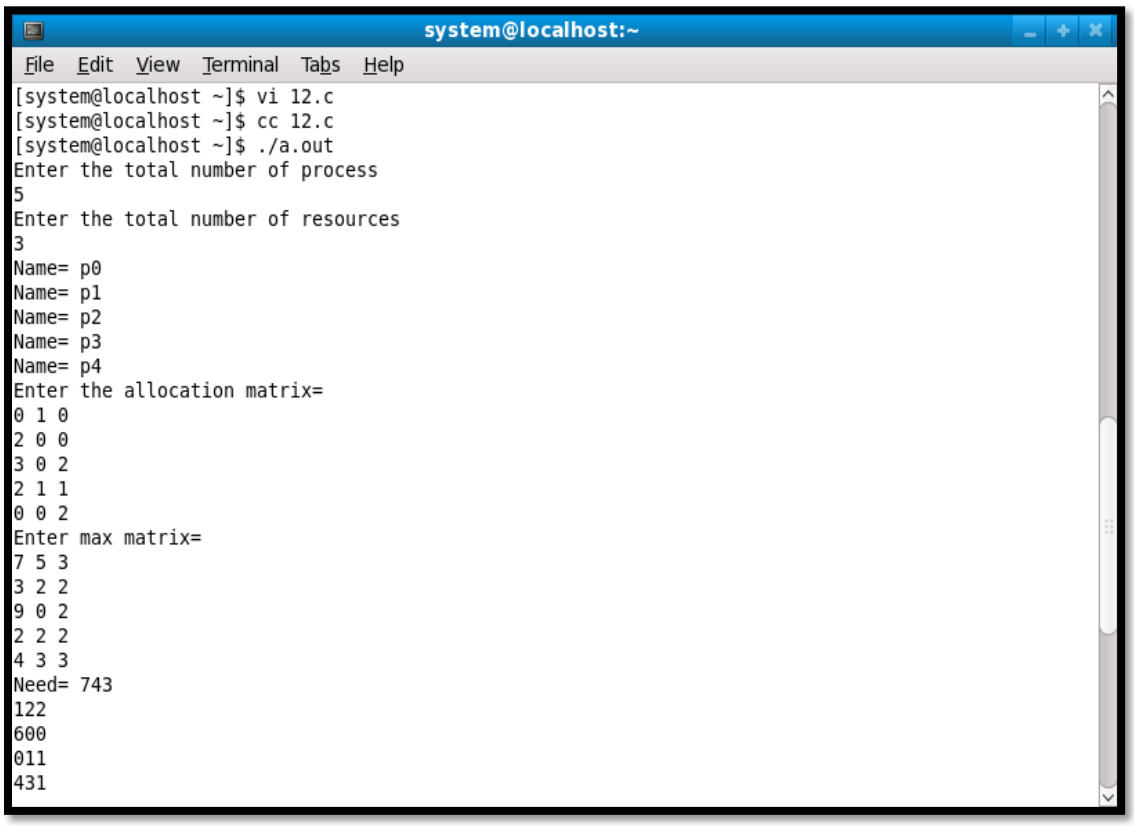
```
#include<stdio.h>
#include<string.h>
int n,m,i,j;
int all[10][10],max[10][10],need[10][10],work[10],work1[10];
struct proc
{
        char name[10];
        int flag;
}pro[20],temp;

void get()
{
        printf("ENTER THE TOTAL NUMBER OF PROCESS:\n");
        scanf("%d",&n);
        printf("ENTER THE TOTAL NUMBER OF RESOURCES:\n");
        scanf("%d",&m);
        for(i=0;i<n;i++)
        {
                printf("\n NAME= ");
                scanf("%s",&pro[i].name);
        }
        printf("ENTER THE ALLOCATION MATRIX= \n");
        for(i=0;i<n;i++)
        for(j=0;j<m;j++)
        scanf("%d",&all[i][j]);
        printf("ENTER MAX MATRIX= \n");
        for(i=0;i<n;i++)
        for(j=0;j<m;j++)
        scanf("%d",&max[i][j]);
        for(i=0;i<n;i++)
        for(j=0;j<m;j++)
        need[i][j]=max[i][j]-all[i][j];
        printf("NEED= ");
        for(i=0;i<n;i++)
        {
```

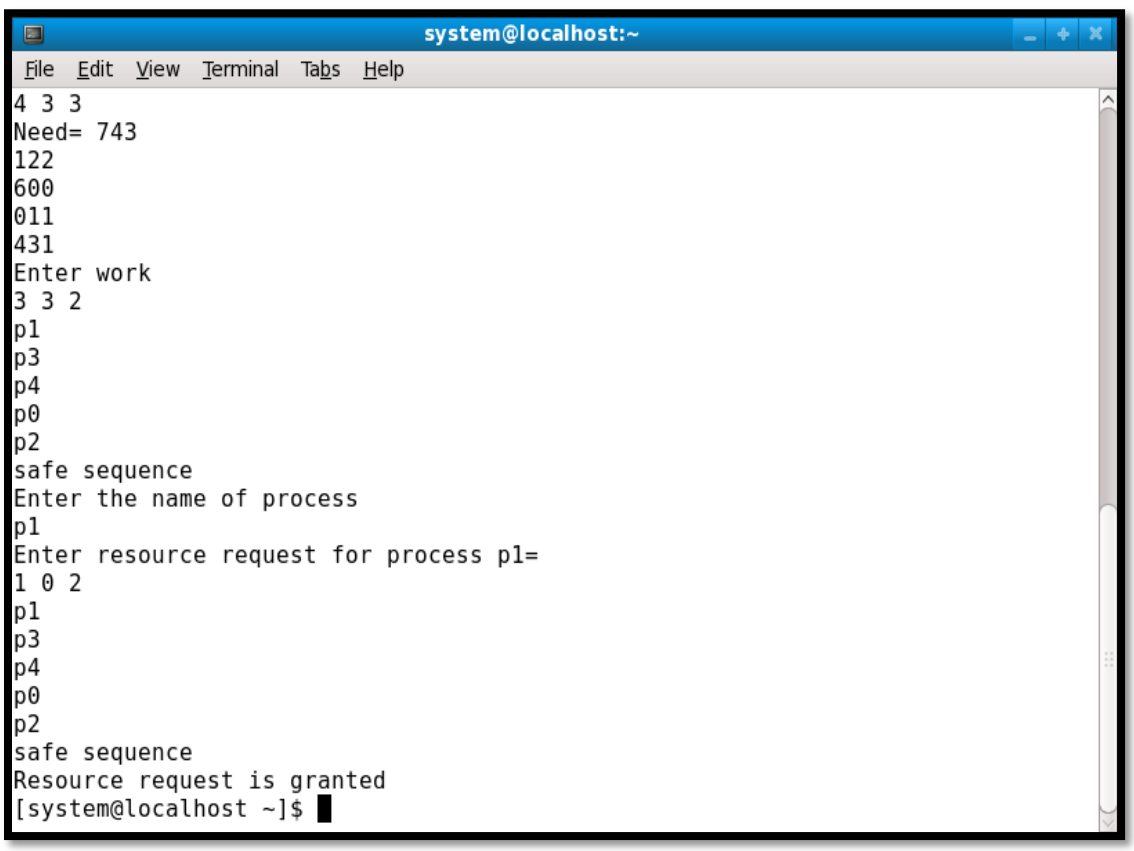
```
                for(j=0;j<m;j++)
                printf("%d",need[i][j]);
                printf("\n");
        }
        printf("ENTER WORK\n");
        for(i=0;i<m;i++)
        {
                scanf("%d",&work[i]);
                work1[i]=work[i];
        }
}

int safe()
{
        int count=0;
        for(i=0;i<n;i++)
        pro[i].flag=0;
        i=0;
        while(1)
        {
                for(j=0;j<m;j++)
                {
                        if(need[i][j]>work[j])
                        break;
                }
                if(pro[i].flag==0  &&  j==m)
                {
                        printf("%s\n",pro[i].name);
                        for(j=0;j<m;j++)
                        work[j]=work[j]+all[i][j];
                        pro[i].flag=1;
                }
                else
                i=++i%n;
                for(j=0;j<n;j++)
                {
                        if(!pro[j].flag)
                        break;
                }
                if(j==n)
                {
                        printf("SAFE SEQUENCE\n");
                        return 1;
```

```
                }

                if(count>100)
                {
                        printf("NOT SAFE SEQUENCE\n");
                        return 0;
                }
        }
}

void ra()
{
        char name[10];
        int flag, rr[10], temp1[10], temp2[10], temp3[10];
        printf("ENTER THE NAME OF PROCESS\n");
        scanf("%s",&name);
        for(i=0;i<n;i++)
        {
                if(!strcmp(name,pro[i].name))
                break;
        }
        printf("ENTER RESOURCE REQUEST FOR PROCESS %s=
        \n",pro[i].name);
        for(j=0;j<m;j++)
        {
                scanf("%d",&rr[j]);
                work[j]=work1[j];
                if(rr[j]>need[i][j])
        {
                printf("REQUEST IS GREATER THAN NEED\n");
                break;
        }
        if(rr[j]>work[j])
        {
                printf("REQUEST IS GREATER THAN AVAILABLE\n");
                break;
        }
}

if(j==m)
{
        for(j=0;j<m;j++)
        {
```

```
                temp1[j]=all[i][j];
                temp2[j]=work[j];
                temp3[j]=need[i][j];
        }
for(j=0;j<m;j++)
{
        all[i][j]=all[i][j]+rr[j];
        need[i][j]=need[i][j]-rr[j];
        work[j]=work[i]-rr[j];
}
flag=safe();
if(flag)
printf("RESOURCE REQUEST IS GRANTED\n");
else
{
        for(j=0;j<m;j++)
        {
                all[i][j]=temp1[j];
                work[j]=temp2[j];
                need[i][j]=temp3[j];
        }
        printf("RESOURCE REQUEST IS NOT GRANTED\n");
   }
  }
}

int main()
{
        get();
        safe();
        ra();
        return(0);
}
```

OUTPUT :


system@localhost:~
File Edit View Terminal Tabs Help
[system@localhost ~]$ vi 12.c
[system@localhost ~]$ cc 12.c
[system@localhost ~]$ ./a.out
Enter the total number of process
5
Enter the total number of resources
3
Name= p0
Name= p1
Name= p2
Name= p3
Name= p4
Enter the allocation matrix=
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter max matrix=
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Need= 743
122
600
011
431


system@localhost:~
File Edit View Terminal Tabs Help
4 3 3
Need= 743
122
600
011
431
Enter work
3 3 2
p1
p3
p4
p0
p2
safe sequence
Enter the name of process
p1
Enter resource request for process p1=
1 0 2
p1
p3
p4
p0
p2
safe sequence
Resource request is granted
[system@localhost ~]$