Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

Program 13: Write a lex program to extract the HTML tags from the input file to the output file.

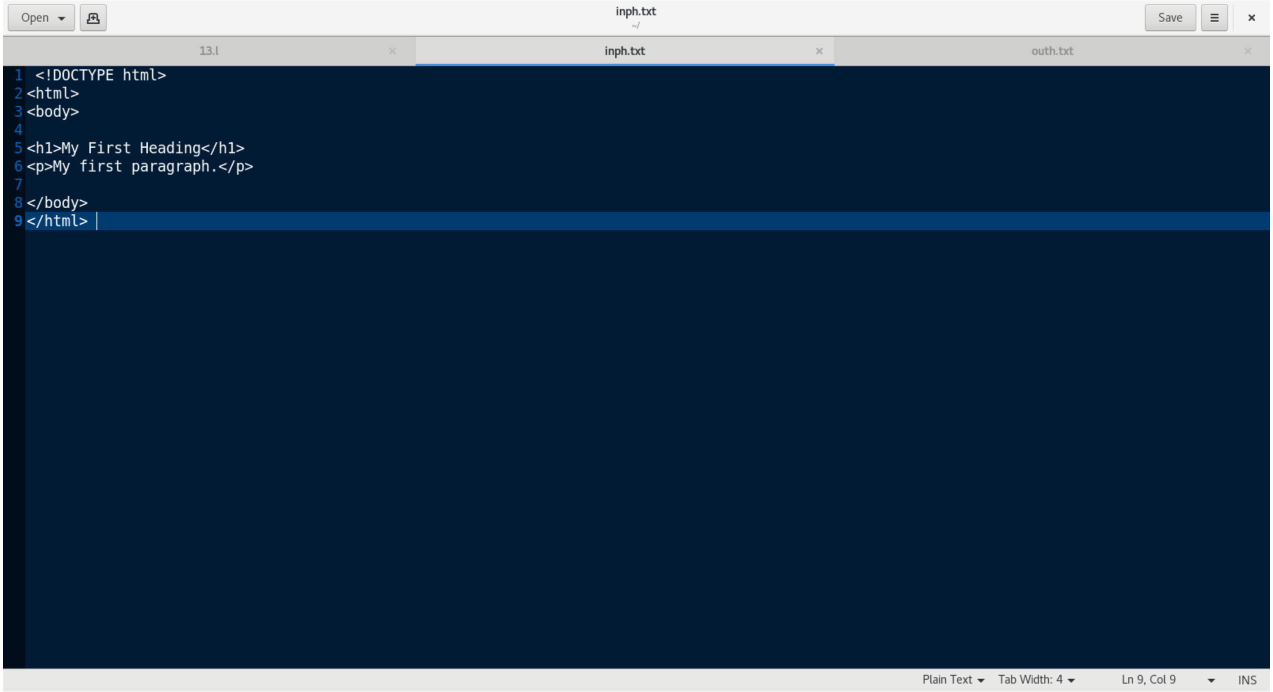
Algorithm-

1. Open the input file (html_input.txt) for reading and the output file (html_output.txt) for writing using yyin and yyout.
2. Rule 1 – Ignore double less-than patterns: Skip tokens that begin with two or more < characters followed by non-space and non-newline characters.
3. Rule 2 – Ignore patterns with unmatched or malformed tags: Skip tokens starting with <, containing slashes or other characters, and ending with two or more > characters.
4. Rule 3 – Match and write valid HTML-like tags: Match properly formed tags like <tag> or </tag> and write them directly to the output file.
5. Default Rule – Ignore all other characters.

```
%{

#include<stdio.h>

%}

%%

[<]{2,}[^\n ]* {};

[<][^//n][/]*[>]{2,} {};

[<][/]?[^<>]*[>] { fprintf(yyout,"%s",yytext);}

. {};

%%

int yywrap(){return 1;}

int main(){

extern FILE *yyin,*yyout;

yyin=fopen("html_input.txt","r");

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

yylex();

return 0;

}
```

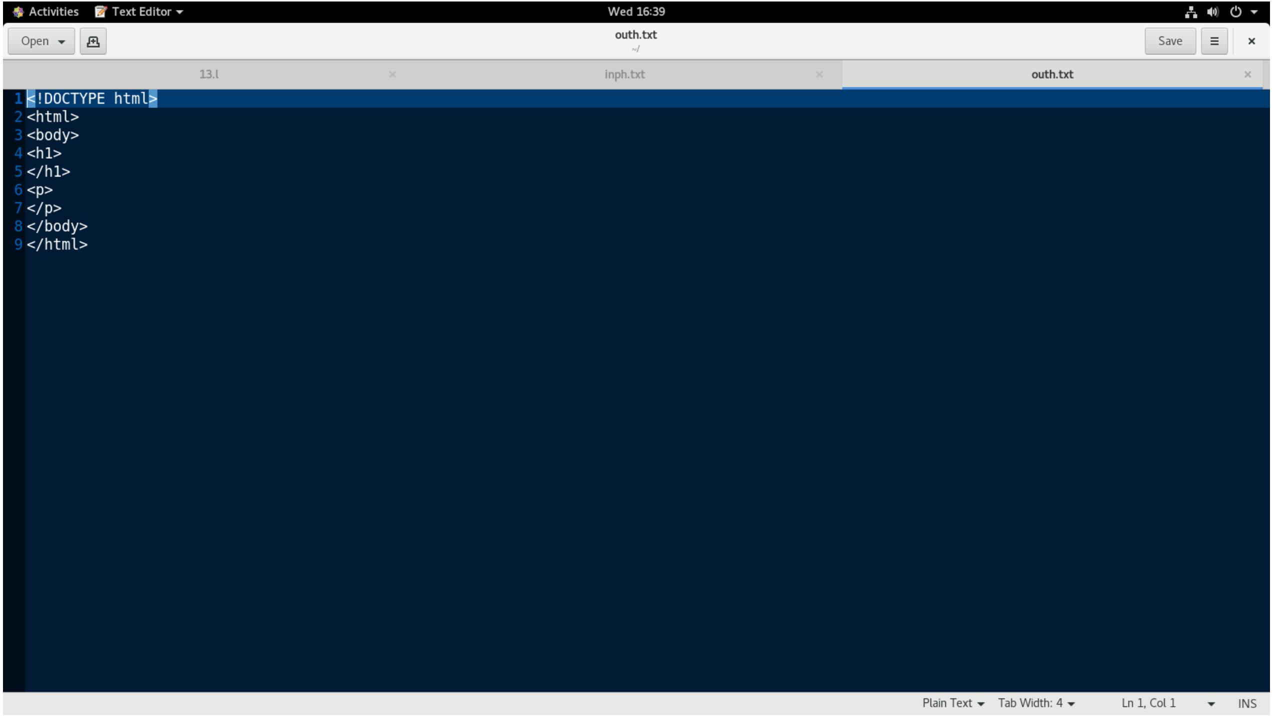
Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

Input:



```
 <!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

Output:



```
<!DOCTYPE html>
<html>
<body>
<h1>
</h1>
<p>
</p>
</body>
</html>
```

Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

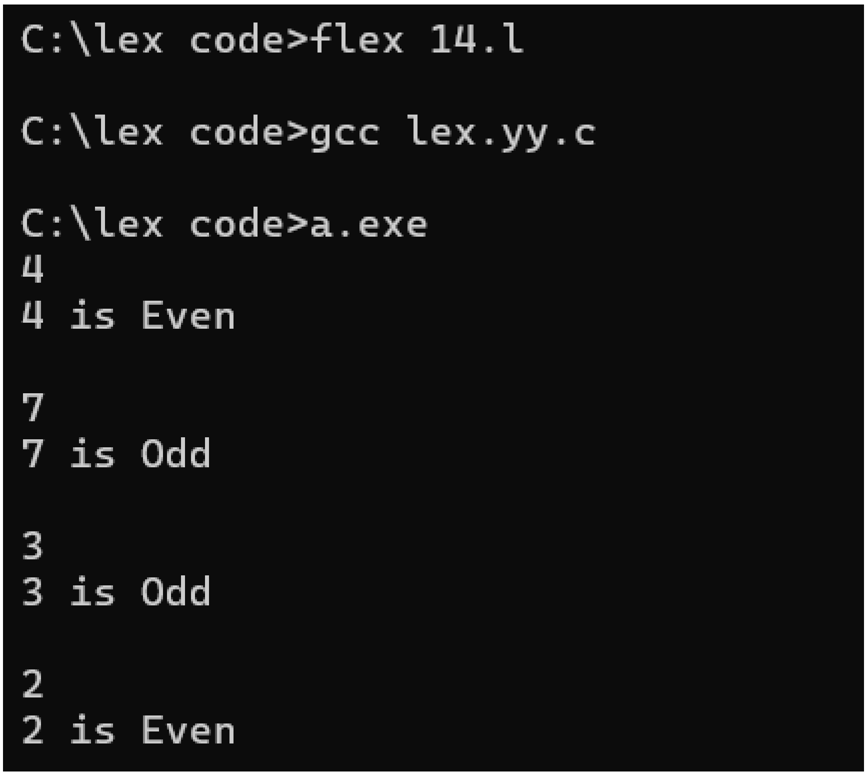
Program 14: Write a lex program to check for even or odd number in input using atoi().

Algorithm-

```
1. Start lexical analysis using yylex() on standard input.
2. Rule 1 – Match numbers ([0-9]+):
    a) Use atoi(yytext) to convert the string to an integer.
    b) If num % 2 == 0: print "Even"
    c) Else: print "Odd"
3. Rule 2 – Ignore others (.|\n):
   Skip non-numeric characters.
4. Define yywrap() to signal end of input.
5. End program after processing all input.
```

```
%{
#include <stdio.h>
#include <stdlib.h>
%}
%%
[0-9]+   {
        int num = atoi(yytext);
        if (num % 2 == 0)
            printf("%d is Even\n", num);
        else
            printf("%d is Odd\n", num);
        }
%%
int yywrap() { return 1; }
int main(){
yylex();
return 0;
}
```

Output:



```
C:\lex code>flex 14.l

C:\lex code>gcc lex.yy.c

C:\lex code>a.exe
4
4 is Even

7
7 is Odd

3
3 is Odd

2
2 is Even
```

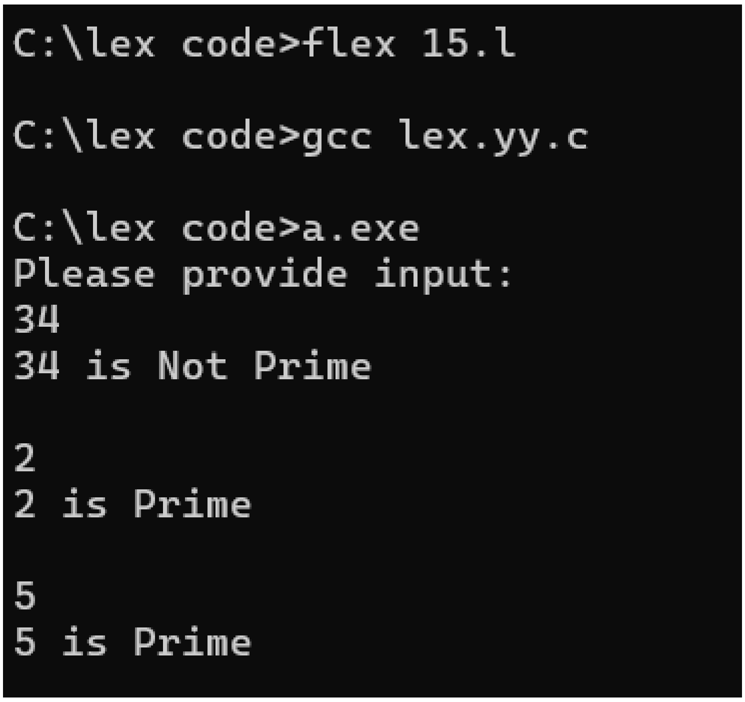
Program 15: Write a lex program to check for prime number in input using atoi().

Algorithm-

1. Include headers: Use stdio.h and stdlib.h for printf() and atoi().
2. Rule 1 – Match integers ([0-9]+): Convert token to integer using atoi().
3. Check prime or not:
    a) If n < 2, it's not prime.
    b) Loop from 2 to n/2. If divisible, return not prime.
    c) Print result.
4. Rule 2 – Ignore non-numeric input (.|\n): Skip all other characters.
5. Define yywrap() and call yylex() in main() to begin tokenizing.

```
%{
#include<stdio.h>
int num, i;
int count=0;
int flag=0;
%}
%%
[0-9]+ {num=atoi(yytext);
        flag =0;
        for(i=2; i<=num/2; i++)
        {
        if(num%i==0)
        {
        printf("Not Prime");
        flag=1;
        break;
        }}
        if(!flag){
        printf("Prime");
        }
        }

%%

int yywrap(){return 0;}

int main(){

printf("Please provide input");

yylex();

return 0;

}
```

Output:



```
C:\lex code>flex 15.l

C:\lex code>gcc lex.yy.c

C:\lex code>a.exe
Please provide input:
34
34 is Not Prime

2
2 is Prime

5
5 is Prime
```

Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

Program 16: Write a lex program to replace the white spaces of an input file with a single blank space and store in the output file.

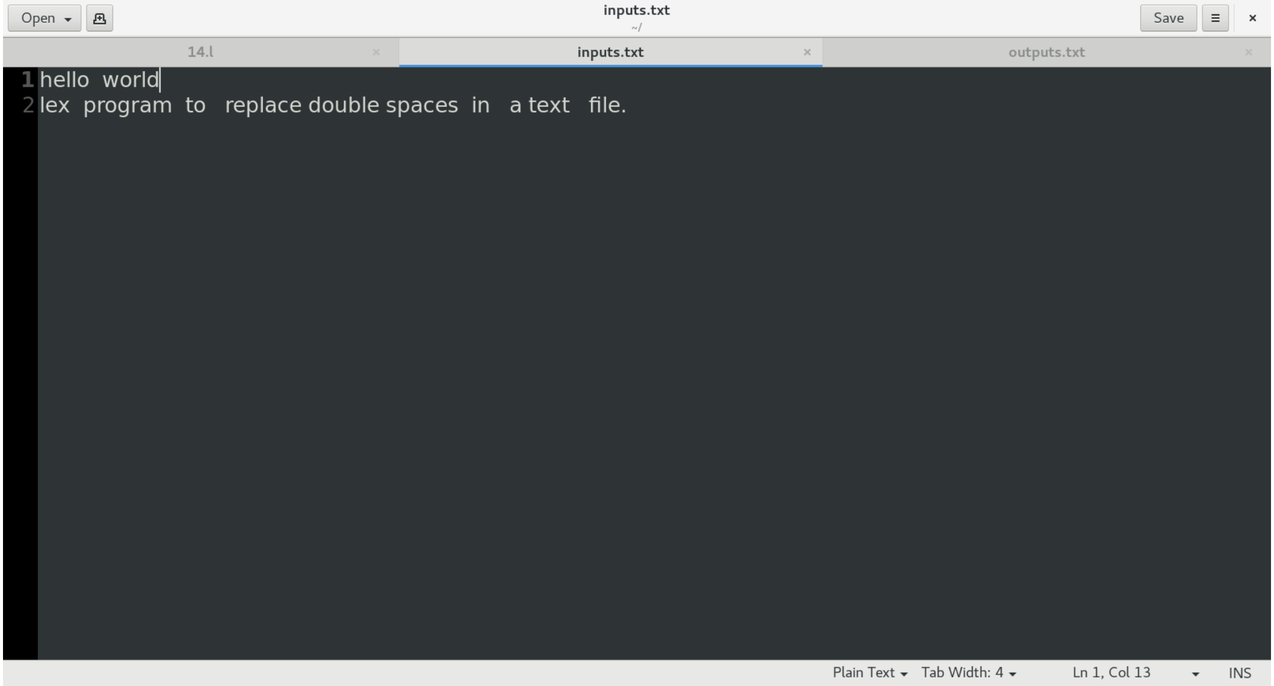
Algorithm-

1. Open input/output files:
   yyin reads from input_space.txt, yyout writes to output_space.txt.
2. Rule 1 – Match spaces or tabs ([\t]+):
   Replace any sequence of spaces/tabs with a single space in output.
3. Rule 2 – Match non-space/tab sequences ([^ \t]+):
   Copy all words or tokens as-is to output.
4. Process input with yylex() and terminate on completion using yywrap().

```
%{

#include<stdio.h>

%}

%%

[ \t]+ {fprintf(yyout," ");}

[^ \t]+ {fprintf(yyout,"%s", yytext);}

%%

int yywrap() {return 1;}

int main(){

extern FILE *yyin, *yyout;

yyin = fopen("input.txt","r");

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

yylex();

return 0;

}
```

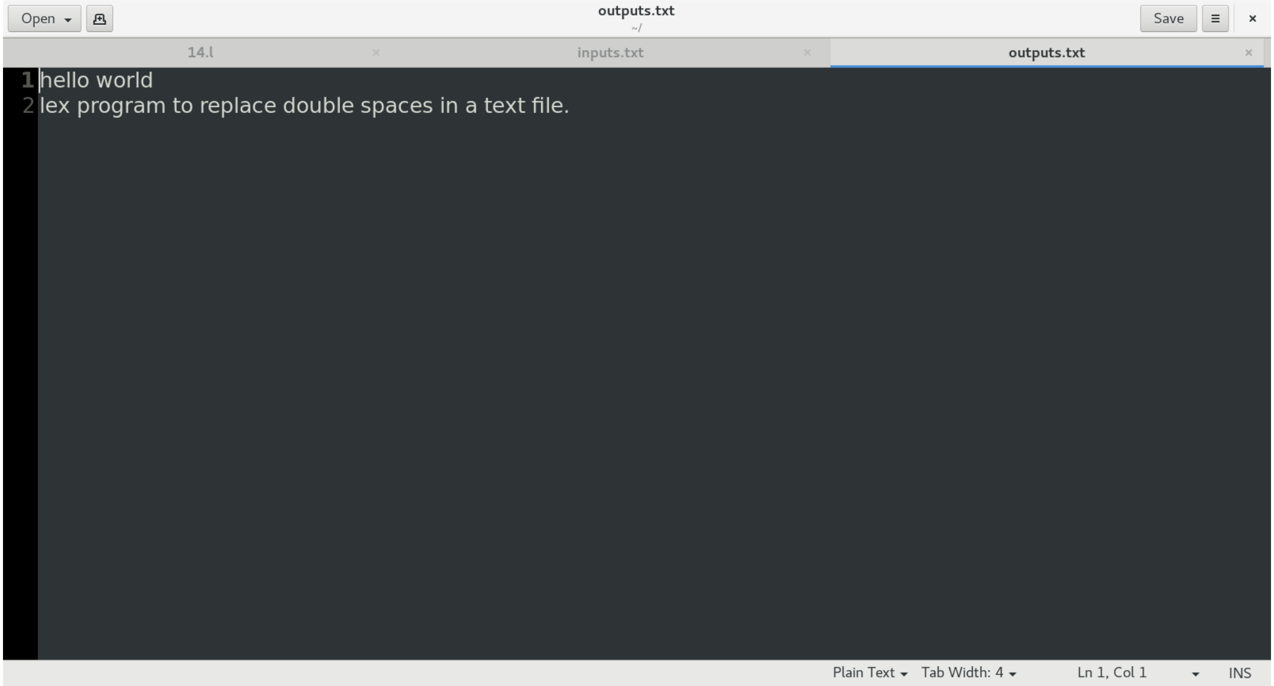
Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

Input:



```
hello  world
lex  program  to   replace double spaces  in   a text   file.
```

Output:



```
hello world
lex program to replace double spaces in a text file.
```

Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

Program 17: Write a lex code to design a DFA that accepts strings ending with 01 over the input characters 0, 1.

DFA design:

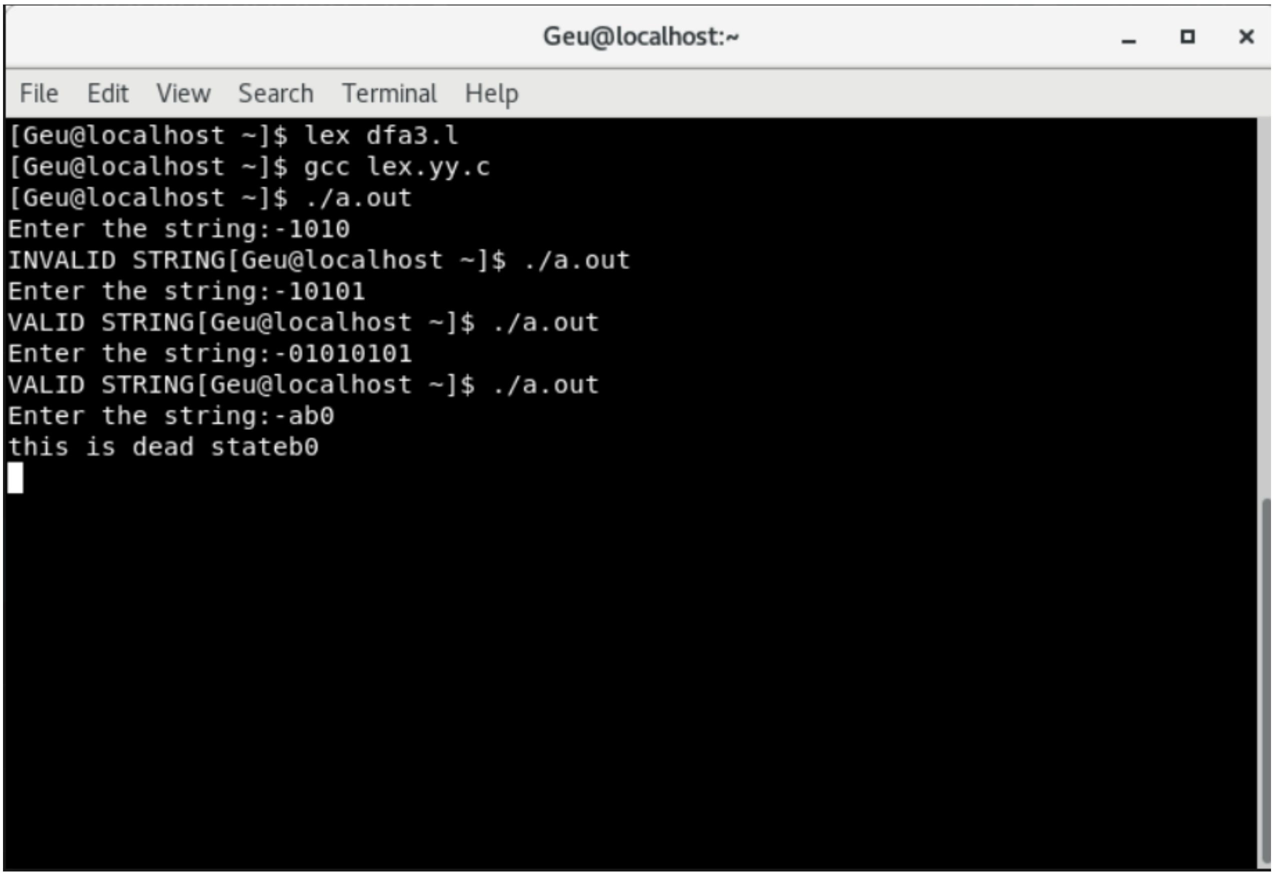
Code:

```
%{

#include<stdio.h>

%}

%s A B DEAD

%%

<INITIAL>0 BEGIN A;

<INITIAL>1 BEGIN INITIAL;

<INITIAL>[^01\n] BEGIN DEAD;{printf("Dead State \n");}

<INITIAL>\n BEGIN INITIAL;{printf("String Rejected \n");}


<A>0 BEGIN A;

<A>1 BEGIN B;

<A>[^01\n] BEGIN DEAD;{printf("Dead State \n");}

<A>\n BEGIN INITIAL;{printf("String Rejected \n");}
```

Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

```
<B>0 BEGIN A;

<B>1 BEGIN INITIAL;

<B>[^01\n] BEGIN DEAD;{printf("Dead State \n");}

<B>\n BEGIN INITIAL;{printf("String Accepted \n");}


<DEAD>.* {printf("Dead State Reached: Program Terminated");}

%%

int yywrap(){return 1;}

int main(){

printf("Enter 0s and 1s: \n");

yylex();

return 0;

}
```

Output:



```
[Geu@localhost ~]$ lex dfa3.l
[Geu@localhost ~]$ gcc lex.yy.c
[Geu@localhost ~]$ ./a.out
Enter the string:-1010
INVALID STRING[Geu@localhost ~]$ ./a.out
Enter the string:-10101
VALID STRING[Geu@localhost ~]$ ./a.out
Enter the string:-01010101
VALID STRING[Geu@localhost ~]$ ./a.out
Enter the string:-ab0
this is dead stateb0
```

Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

Program 18: Write a lex code to design a DFA that accepts strings ending with b over the input characters a, b.

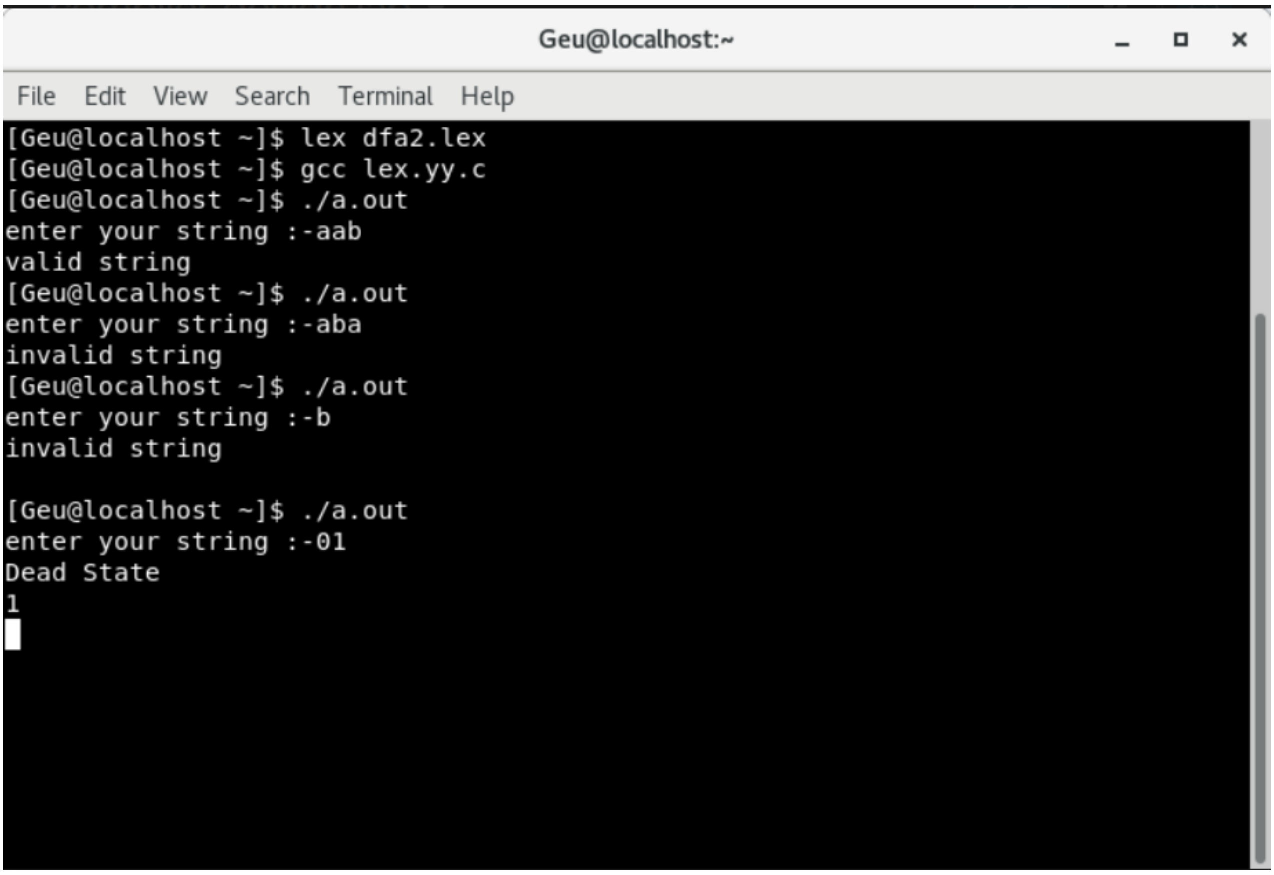
DFA design:

Code:

```
%{
#include<stdio.h>
%}
%s A DEAD
%%
<INITIAL>a BEGIN INITIAL;
<INITIAL>b BEGIN A;
<INITIAL>[^ab\n] BEGIN DEAD;{printf("Dead State \n");}
<INITIAL>\n BEGIN INITIAL;{printf("String Rejected \n");}

<A>a BEGIN INITIAL;
<A>b BEGIN A;
<A>[^ab\n] BEGIN DEAD;{printf("Dead State \n");}
<A>\n BEGIN A;{printf("String Accepted \n");}
```

```
<DEAD>. {printf("String Rejected \n");}

%%

int yywrap(){return 1;}

int main(){

printf("Enter only a's and b's : \n");

yylex();

return 0;

}
```

Output:



```
[Geu@localhost ~]$ lex dfa2.lex
[Geu@localhost ~]$ gcc lex.yy.c
[Geu@localhost ~]$ ./a.out
enter your string :-aab
valid string
[Geu@localhost ~]$ ./a.out
enter your string :-aba
invalid string
[Geu@localhost ~]$ ./a.out
enter your string :-b
invalid string

[Geu@localhost ~]$ ./a.out
enter your string :-01
Dead State
1
```

Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

Program 19: Write a lex code to design a DFA that accepts strings that start with a and end with b over the input characters a, b.

DFA design:

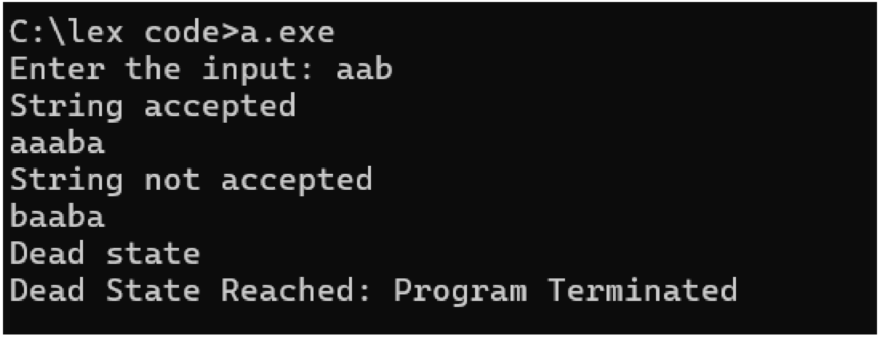
Code:

```
%{
#include<stdio.h>
%}
%s A B DEAD
%%
<INITIAL>a BEGIN A;
<INITIAL>b BEGIN DEAD;{printf("Dead state\n");}
<INITIAL>[^ab\n] BEGIN DEAD;{printf("Dead state\n");}
<INITIAL>\n BEGIN INITIAL; {printf("String not accepted\n");}

<A>a BEGIN A;
<A>b BEGIN B;
<A>[^ab\n] BEGIN DEAD;{printf("Dead state\n");}
<A>\n BEGIN A;{printf("String not accepted\n");}
```

Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

```
<B>a BEGIN A;

<B>b BEGIN B;

<B>[^ab\n] BEGIN DEAD;{printf("Dead state\n");}

<B>\n BEGIN B;{printf("String accepted\n");}


<DEAD>.* BEGIN DEAD; {printf("Dead State Reached: Program Terminated\n");}

%%

int yywrap(){return 1;}

int main()

{

   printf("Enter the input : ");

   yylex();

   return 0;

}
```

Output:



```
C:\lex code>a.exe
Enter the input: aab
String accepted
aaaba
String not accepted
baaba
Dead state
Dead State Reached: Program Terminated
```

Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

Program 20: Write a lex code to design a DFA that accepts strings with even number of 0s and even number of 1s over the input characters 0, 1.

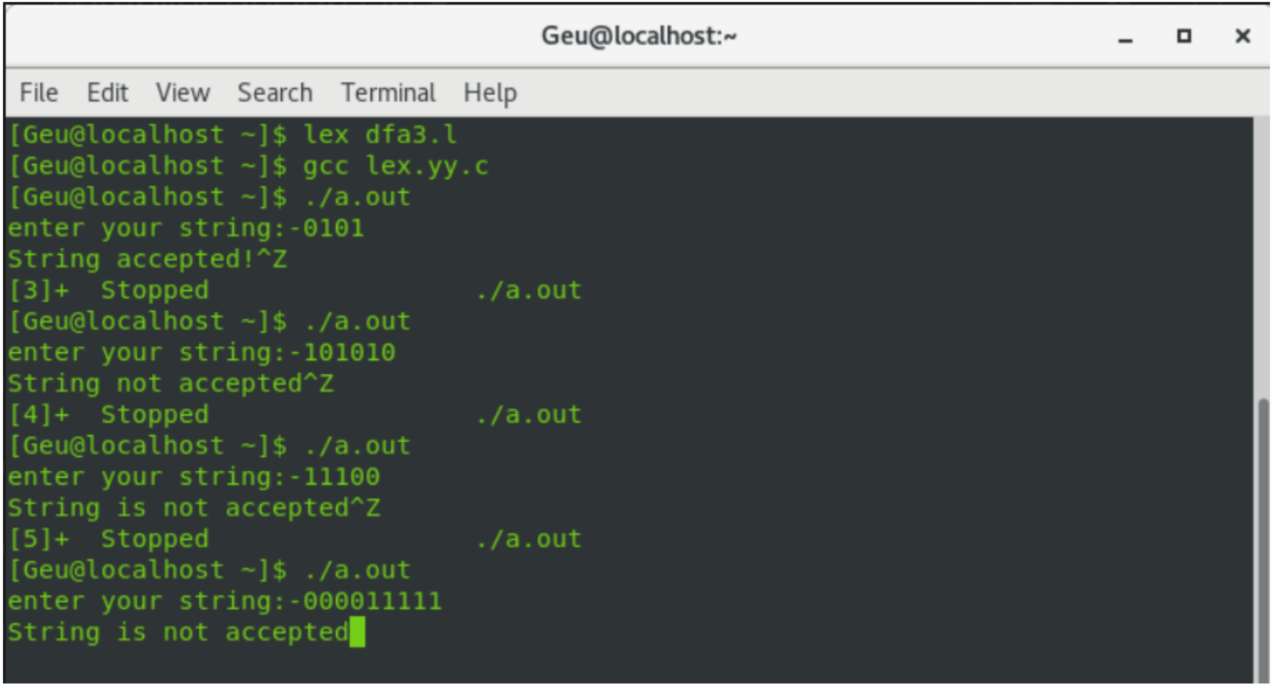
DFA design:

Code;

```
%{
#include<stdio.h>
%}
%s A B C DEAD
%%
<INITIAL>0 BEGIN B;
<INITIAL>1 BEGIN A;
<INITIAL>[^01\n] BEGIN DEAD; {printf("Dead State\n");}
<INITIAL>\n BEGIN INITIAL; {printf("String Accepted\n");}

<A>0 BEGIN C;
<A>1 BEGIN INITIAL;
<A>[^01\n] BEGIN DEAD; {printf("Dead State\n");}
<A>\n BEGIN INITIAL; {printf("String Rejected\n");}
```

```
<B>0 BEGIN INITIAL;

<B>1 BEGIN C;

<B>[^01\n] BEGIN DEAD; {printf("Dead State\n");}

<B>\n BEGIN INITIAL; {printf("String Rejected\n");}


<C>0 BEGIN A;

<C>1 BEGIN B;

<C>[^01\n] BEGIN DEAD; {printf("Dead State\n");}

<C>\n BEGIN INITIAL; {printf("String Rejected\n");}


<DEAD>.* {printf("Dead State: Program Terminated\n");}

%%

int yywrap(){return 1;}

int main() {

   printf("Enter only 0s and 1s: ");

   yylex();

   return 0;

}
```

Output:



```
[Geu@localhost ~]$ lex dfa3.l
[Geu@localhost ~]$ gcc lex.yy.c
[Geu@localhost ~]$ ./a.out
enter your string:-0101
String accepted!^Z
[3]+  Stopped                 ./a.out
[Geu@localhost ~]$ ./a.out
enter your string:-101010
String not accepted^Z
[4]+  Stopped                 ./a.out
[Geu@localhost ~]$ ./a.out
enter your string:-11100
String is not accepted^Z
[5]+  Stopped                 ./a.out
[Geu@localhost ~]$ ./a.out
enter your string:-000011111
String is not accepted
```

Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

Program 21: Write a lex code to design a DFA that accepts a string with odd number of 0s and odd number of 1s over the input characters 0,1.

DFA design:

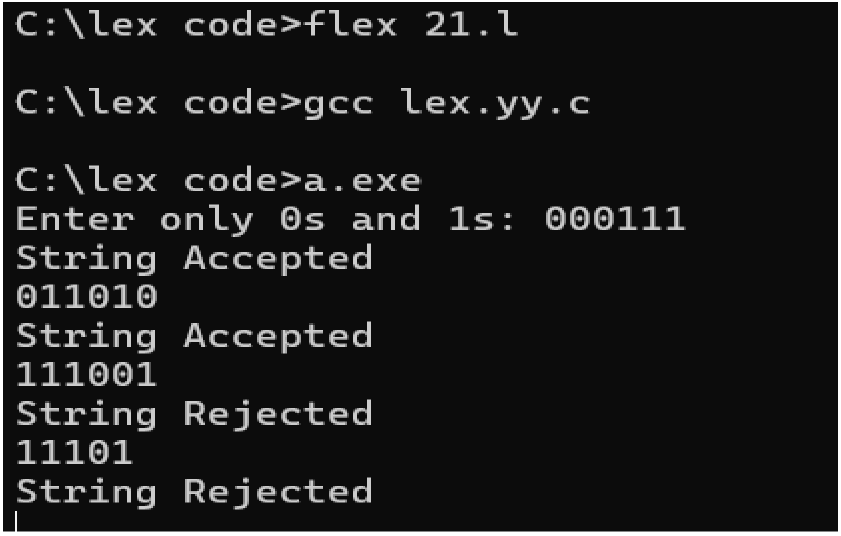
Code:

```
%{
#include<stdio.h>
%}
%s A B C DEAD
%%
<INITIAL>0 BEGIN B;
<INITIAL>1 BEGIN A;
<INITIAL>[^01\n] BEGIN DEAD; {printf("Dead State\n");}
<INITIAL>\n BEGIN INITIAL; {printf("String Rejected\n");}

<A>0 BEGIN C;
<A>1 BEGIN INITIAL;
<A>[^01\n] BEGIN DEAD; {printf("Dead State\n");}
<A>\n BEGIN INITIAL; {printf("String Rejected\n");}
```

Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

```
<B>0 BEGIN INITIAL;

<B>1 BEGIN C;


<B>[^01\n] BEGIN DEAD; {printf("Dead State\n");}

<B>\n BEGIN INITIAL; {printf("Sring Rejected\n");}


<C>0 BEGIN A;

<C>1 BEGIN B;

<C>[^01\n] BEGIN DEAD; {printf("Dead State\n");}

<C>\n BEGIN INITIAL; {printf("String Accepted\n");}

<DEAD>. {printf("Dead State: Program terminated\n");}

%%

int yywrap(){return 1;}

int main()

{

   printf("Enter only 0s and 1s: ");

   yylex();

   return 0;

}
```

Output:



```
C:\lex code>flex 21.l

C:\lex code>gcc lex.yy.c

C:\lex code>a.exe
Enter only 0s and 1s: 000111
String Accepted
011010
String Accepted
111001
String Rejected
11101
String Rejected
```

Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

Program 22: Write a lex code to design a DFA that accepts strings ending with 0011 over the input characters 0,1.

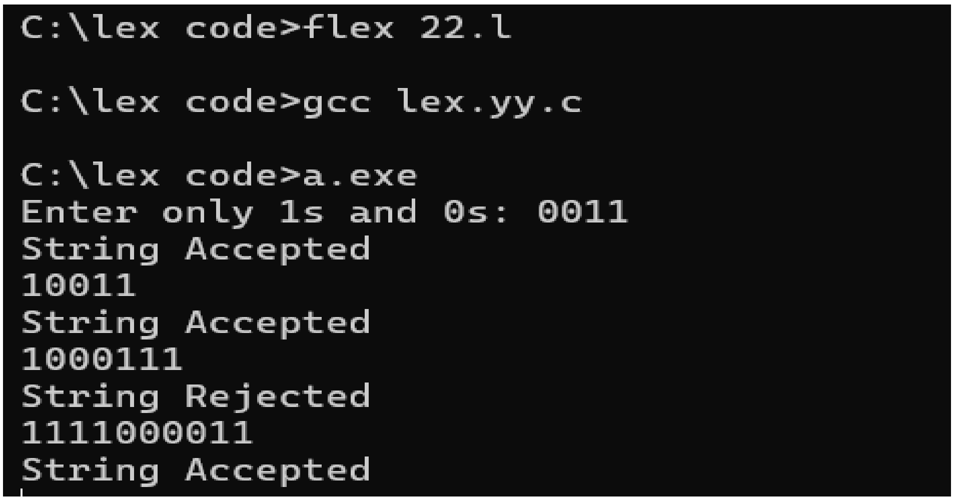
DFA design:

Code:

```
%{

#include<stdio.h>

%}

%s A B C D DEAD

%%

<INITIAL>0 BEGIN A;

<INITIAL>1 BEGIN INITIAL;

<INITIAL>[^01\n] BEGIN DEAD;{printf("Dead State\n");}

<INITIAL>\n {printf("String Rejected\n");}


<A>0 BEGIN B;

<A>1 BEGIN INITIAL;

<A>[^01\n] BEGIN DEAD;{printf("Dead State\n");}

<A>\n {printf("String Rejected\n");}


<B>0 BEGIN B;

<B>1 BEGIN C;
```

```
<B>[^01\n] BEGIN DEAD;{printf("Dead State");}

<B>\n {printf("String Rejected\n");}


<C>0 BEGIN A;

<C>1 BEGIN D;

<C>[^01\n] BEGIN DEAD;{printf("Dead State\n");}

<C>\n {printf("String Rejected\n");}


<D>0 BEGIN A;

<D>1 BEGIN INITIAL;

<D>[^01\n] BEGIN DEAD;{printf("Dead State\n");}

<D>\n {printf("String Accepted\n");}


<DEAD>.* {printf("Dead State\n");}

%%

int yywrap(){return 1;}

int main(){

printf("Enter only 1s and 0s:");

yylex();

return 0;

}
```

Output:



```
C:\lex code>flex 22.l

C:\lex code>gcc lex.yy.c

C:\lex code>a.exe
Enter only 1s and 0s: 0011
String Accepted
10011
String Accepted
1000111
String Rejected
1111000011
String Accepted
```

Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

Program 23: Write a lex code to design a DFA that accepts strings containing three consecutive 0s, over the input characters 0,1.

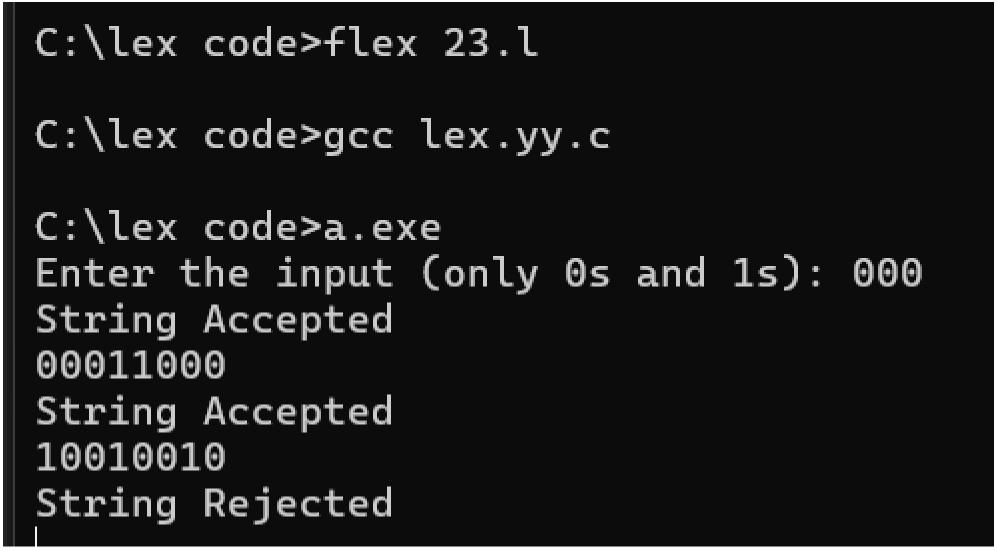
DFA design:

Code:

```
%{
#include<stdio.h>
%}
%s A B C DEAD
%%
<INITIAL>0 BEGIN A;
<INITIAL>1 BEGIN INITIAL;
<INITIAL>[^01\n] BEGIN DEAD; {printf("Dead State\n");}
<INITIAL>\n {printf("String Rejected\n");}

<A>0 BEGIN B;
<A>1 BEGIN INITIAL;
<A>[^01\n] BEGIN DEAD;{printf("Dead State\n");}
<A>\n {printf("String Rejected\n");}

<B>0 BEGIN C;
```

```
<B>1 BEGIN INITIAL;

<B>[^01\n] BEGIN DEAD;{printf("Dead State\n");}

<B>\n {printf("String Rejected\n");}


<C>0 BEGIN C;

<C>1 BEGIN C;

<C>[^01\n] BEGIN DEAD;{printf("Dead State\n");}

<C>\n BEGIN INITIAL;{printf("String Accepted\n");}


<DEAD>.* {printf("Dead State: Program Terminated\n");}

%%

int yywrap(){return 1;}

int main()

{

   printf("Enter the input (only 0s and 1s): ");

   yylex();

   return 0;

}
```

Output:



```
C:\lex code>flex 23.l

C:\lex code>gcc lex.yy.c

C:\lex code>a.exe
Enter the input (only 0s and 1s): 000
String Accepted
00011000
String Accepted
10010010
String Rejected
```

Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

Program 24: Write a lex program that accepts a string with the third last symbol as 0 over the input characters 0, 1.

DFA design:

Code:

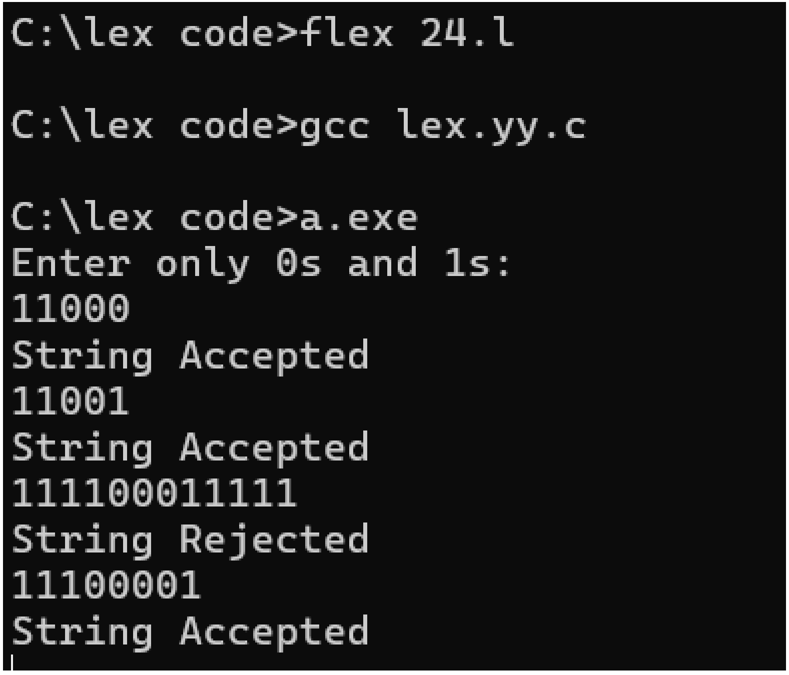
```
%{
#include<stdio.h>
%}
%s A B C D E F G DEAD
%%
<INITIAL>0 BEGIN A;
<INITIAL>1 BEGIN INITIAL;
<INITIAL>[^01\n] BEGIN DEAD; {printf("Dead State\n");}
<INITIAL>\n {printf("String Rejected\n");}

<A>0 BEGIN B;
<A>1 BEGIN F;
<A>[^01\n] BEGIN DEAD;{printf("Dead State\n");}
<A>\n {printf("String Rejected\n");}
```

```
<B>0 BEGIN C;
<B>1 BEGIN D;
<B>[^01\n] BEGIN DEAD;{printf("Dead State\n");}
<B>\n {printf("String Rejected\n");}

<C>0 BEGIN C;
<C>1 BEGIN D;
<C>[^01\n] BEGIN DEAD;{printf("Dead State\n");}
<C>\n {printf("String Accepted\n");}

<D>0 BEGIN E;
<D>1 BEGIN G;
<D>[^01\n] BEGIN DEAD;{printf("Dead State\n");}
<D>\n {printf("String Accepted\n");}

<E>0 BEGIN B;
<E>1 BEGIN F;
<E>[^01\n] BEGIN DEAD;{printf("Dead State\n");}
<E>\n {printf("String Accepted\n");}

<F>0 BEGIN E;
<F>1 BEGIN G;
<F>[^01\n] BEGIN DEAD;{printf("Dead State\n");}
<F>\n {printf("String Rejected\n");}

<G>0 BEGIN A;
<G>1 BEGIN INITIAL;
<G>[^01\n] BEGIN DEAD;{printf("Dead State\n");}
<G>\n {printf("String Accepted\n");}

<DEAD>.* {printf("Dead State: Program terminated\n");}
```

Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

```
%%

int yywrap(){return 1;}

int main(){

printf("Enter only 0s and 1s");

yylex();

return 0;

}
```

Output:



```
C:\lex code>flex 24.l

C:\lex code>gcc lex.yy.c

C:\lex code>a.exe
Enter only 0s and 1s:
11000
String Accepted
11001
String Accepted
111100011111
String Rejected
11100001
String Accepted
```

Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

Program 25: Write a YACC program to recognize strings in the given grammar:

{ a^nb; n >= 0} .

Algorithm:

Lex Code:

- Match input symbol 'a' and return token A.
- Match input symbol 'b' and return token B.
- Match newline \n and return it to trigger parse check.

YACC Code:

- Start rule is S '\n', which ensures the expression ends correctly.
- Grammar:
  - S -> B: Base case, no as.
  - S -> A S: For every ‘a’, expect another ‘S’, eventually ending with one ‘B’.

Action:

- If input string follows the pattern (any number of a’s followed by exactly one ‘b’), print "String Accepted".
- On any parse error, print "String not Accepted" and terminate.

Lex Code:

```
%{

#include "y.tab.h"

%}


%%

a {return A;}

b {return B;}

[\n] {return '\n';}

%%
```

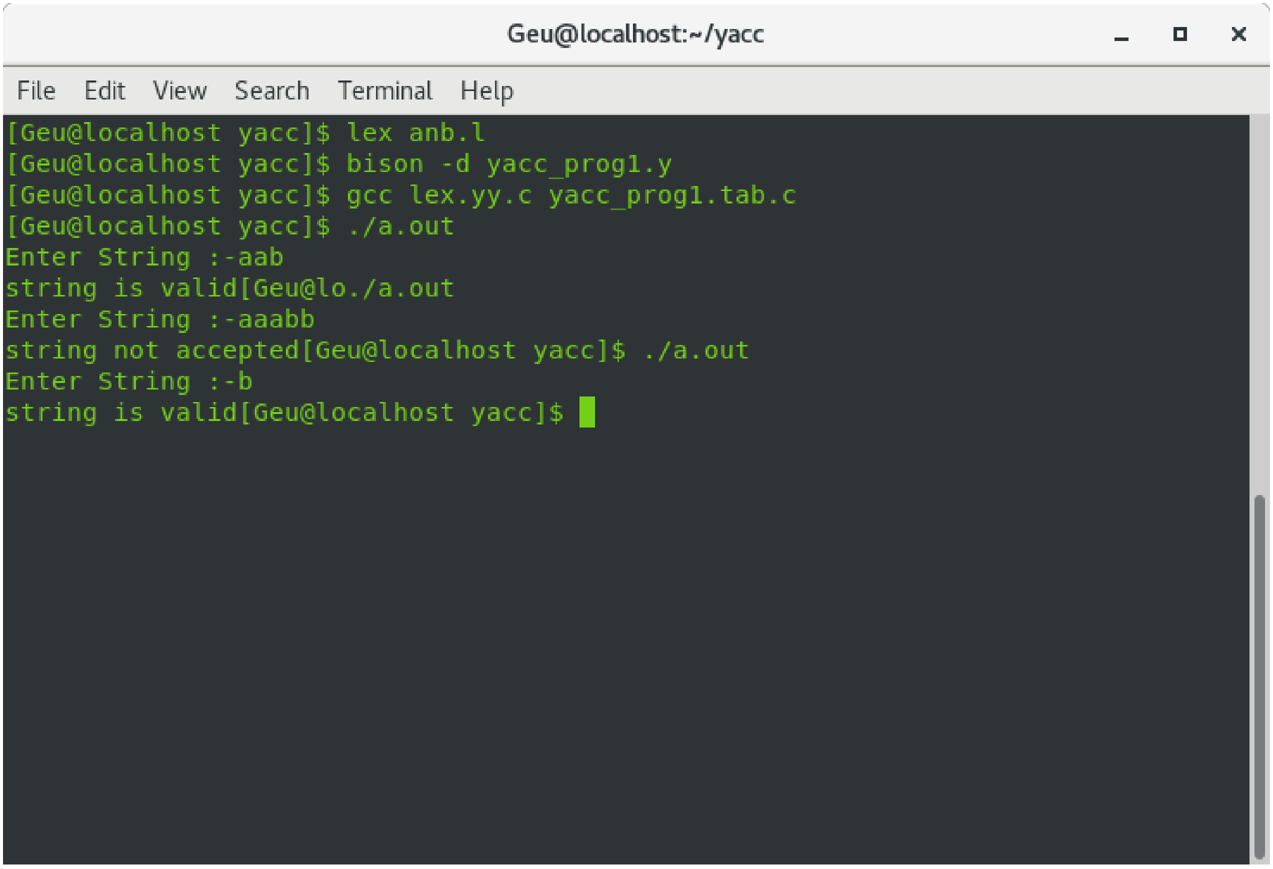
YACC Code:

```
%{

#include<stdio.h>
```

Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

```
#include<stdlib.h>

int yyerror();

int yylex();

%}


%token A B


%%

start: S '\n' {printf("String Accepted \n"); exit(0); }

   S: B | A S ;

%%


int main(){

printf("Enter a String \n");

yyparse();

return 1;

}


int yyerror()

{

printf("String not Accepted \n");

exit(0);

}


int yywrap(){

return 1;

}
```

Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)



```
[Geu@localhost yacc]$ lex anb.l
[Geu@localhost yacc]$ bison -d yacc_prog1.y
[Geu@localhost yacc]$ gcc lex.yy.c yacc_prog1.tab.c
[Geu@localhost yacc]$ ./a.out
Enter String :-aab
string is valid[Geu@lo./a.out
Enter String :-aaabb
string not accepted[Geu@localhost yacc]$ ./a.out
Enter String :-b
string is valid[Geu@localhost yacc]$ 
```

Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

Program 26: Write a YACC program to accept strings in the given grammar:

{ a^nb^n ; n >= 0 }.

Algorithm;

Lex Code:

- Return A for 'a', B for 'b', and newline to trigger parse validation.

YACC Code:

- Start rule: S '\n'.
- Recursive production:
    - S -> A S B: For each ‘a’, match one ‘b’ later.
    - S -> ε: Base case allows empty string (n = 0).

Action:

- If every ‘a’ is matched by a ‘b’ in correct order, print "String is valid".
- Otherwise, output "String not accepted".

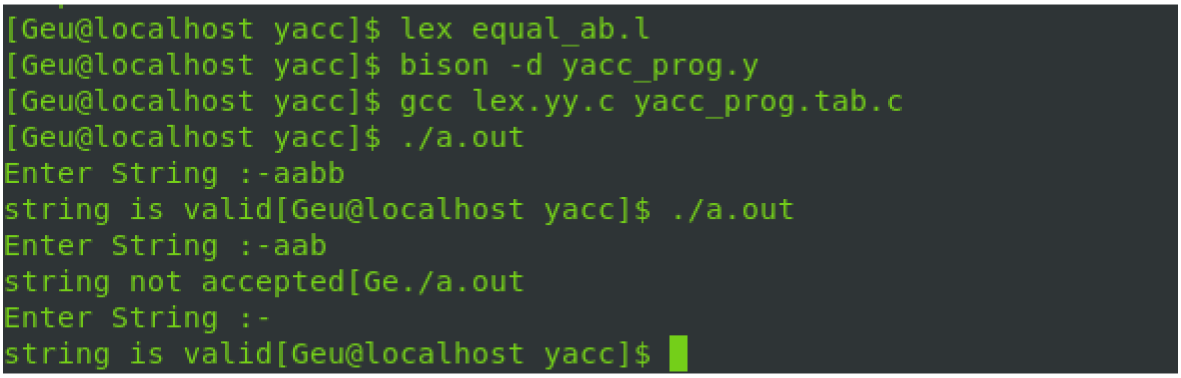
Lex Code:

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

%%
a {return A;}
b {return B;}
[\n] {return '\n';}
%%
```

YACC Code:

```
%{
  #include<stdio.h>
  #include<stdlib.h>
  int yylex();
```

```
    int yyerror();
%}
%token A B


%%
start: S '\n' {printf("String is valid \n"); exit(0);}
    S: A S B |;
%%


int main()
{
    printf("Enter a string \n");
    yyparse();
    return 1;
}


int yyerror()
{
    printf("String not accepted \n");
    exit(0);
}
int yywrap()
{
    return 1;
}
```



```
[Geu@localhost yacc]$ lex equal_ab.l
[Geu@localhost yacc]$ bison -d yacc_prog.y
[Geu@localhost yacc]$ gcc lex.yy.c yacc_prog.tab.c
[Geu@localhost yacc]$ ./a.out
Enter String :-aabb
string is valid[Geu@localhost yacc]$ ./a.out
Enter String :-aab
string not accepted[Ge./a.out
Enter String :-
string is valid[Geu@localhost yacc]$ 
```

Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)

Program 27: Write a YACC program to recognize valid arithmeric expressions with the operators: * , + , / , - , (,)

Algorithm:

Lex Code:

- Return A for 'a', B for 'b', and newline to trigger parse validation.

YACC Code:

- Start rule: S '\n'.
- Recursive production:
  - S -> A S B: For each a, match one b later.
  - S -> ε: Base case allows empty string (n = 0).

Action:

- If every a is matched by a b in correct order, print "String is valid".
- Otherwise, output "String not accepted".

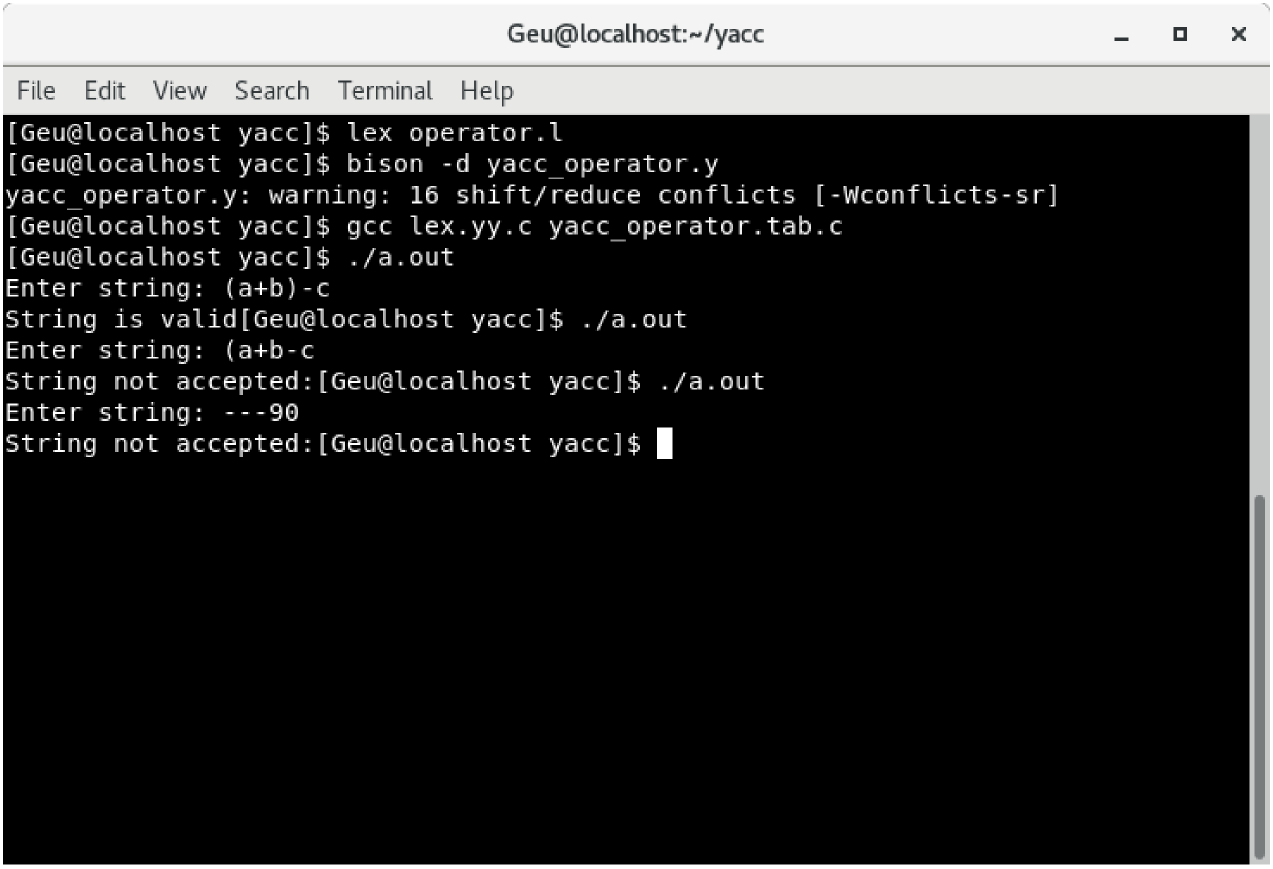
Lex Code:

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

%%
[a-zA-Z] {return ALPHA;}
[0-9]+ {return NUMBER;}
[\n] {return '\n';}
[\t]+ ;
. {return yytext[0];}
%%
```

YACC Code:

```
%{
#include<stdio.h>
#include<stdlib.h>
int yylex();
int yyerror();
%}

%token ALPHA NUMBER
%left '/' '*'
%left '+' '-'
%%
start: S '\n' {printf("Expression Accepted\n"); exit(0);}
        S: S '/' S | S '*' S | S '+' S | S '-' S | '(' S ')' | ALPHA | NUMBER;
%%

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

int yyerror(){
printf("Expression not accepted\n");
exit(0);
}

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

Name:-Divakar Pandey
Section:-DS1
Roll No.:42(2021682)



```
[Geu@localhost yacc]$ lex operator.l
[Geu@localhost yacc]$ bison -d yacc_operator.y
yacc_operator.y: warning: 16 shift/reduce conflicts [-Wconflicts-sr]
[Geu@localhost yacc]$ gcc lex.yy.c yacc_operator.tab.c
[Geu@localhost yacc]$ ./a.out
Enter string: (a+b)-c
String is valid[Geu@localhost yacc]$ ./a.out
Enter string: (a+b-c
String not accepted:[Geu@localhost yacc]$ ./a.out
Enter string: ---90
String not accepted:[Geu@localhost yacc]$ 
```