Q9. Design a DFA in LEX Code which accepts string containing even number of 'a' and even number of 'b' over input alphabet {a, b}.

Code:

```
%{
%}

%s A B

%%
<INITIAL>a BEGIN INITIAL;
<INITIAL>b BEGIN A;
<INITIAL>[^0|\n] BEGIN B;
<INITIAL>\n BEGIN INITIAL; printf("Accepted\n");
<A>a BEGIN A;
<A>b BEGIN INITIAL;
<A>[^0|\n] BEGIN B;
<A>\n BEGIN INITIAL; printf("Not Accepted\n");
<B>b BEGIN B;
<B>a BEGIN B;
<B>[^0|\n] BEGIN B;
<B>\n {BEGIN INITIAL; printf("INVALID\n");}
%%

voidmain()
{
yylex();
}
```

Output:

```
abbb
Not Accepted
babababa
Accepted
```

Q10. Design a DFA in LEX Code which accepts string containing third last element 'a' over input alphabet {a, b}.

Code:

```
%{
%}
%s A B C D E F G DEAD
%%
<INITIAL>b BEGIN INITIAL;
<INITIAL>a BEGIN A;
<INITIAL>[^ab\n] BEGIN DEAD;
<INITIAL>\n BEGIN INITIAL; {printf("Not Accepted\n");}

<A>b BEGIN F;
<A>a BEGIN B;
<A>[^ab\n] BEGIN DEAD;
<A>\n BEGIN INITIAL; {printf("Not Accepted\n");}

<B>b BEGIN D;
<B>a BEGIN C;
<B>[^ab\n] BEGIN DEAD;
<B>\n BEGIN INITIAL; {printf("Not Accepted\n");}

<C>b BEGIN D;
<C>a BEGIN C;
<C>[^ab\n] BEGIN DEAD;
<C>\n BEGIN INITIAL; {printf("Accepted\n");}

<D>b BEGIN G;
<D>a BEGIN E;
<D>[^ab\n] BEGIN DEAD;
<D>\n BEGIN INITIAL; {printf("Accepted\n");}

<E>b BEGIN F;
<E>a BEGIN B;
<E>[^ab\n] BEGIN DEAD;
<E>\n BEGIN INITIAL; {printf("Accepted\n");}

<F>b BEGIN G;
<F>a BEGIN E;
<F>[^ab\n] BEGIN DEAD;
```

```
<F>\n BEGIN INITIAL; {printf("Not Accepted\n");}

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

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

%%
int yywrap()
{
 return 1;
}
int main()
{
 printf("Enter String\n");
 yylex();
 return 0;
}
```

Output:

```
Enter String
aaab
Accepted
ab
Not Accepted
ababab
Not Accepted
123
Invalid
bbba
Not Accepted
bbbaaab
Accepted
```

Q11. Design a DFA in LEX Code for string ending with “11”.
Code:

```
%{

%}
%s A B DEAD

%%
<INITIAL>1 BEGIN A;
<INITIAL>0 BEGIN INITIAL;
<INITIAL>[^01\n] BEGIN DEAD;
<INITIAL>\n BEGIN INITIAL; {printf("Not Accepted\n");}

<A>1 BEGIN B;
<A>0 BEGIN INITIAL;
<A>[^01\n] BEGIN DEAD;
<A>\n BEGIN INITIAL; {printf("Not Accepted\n");}

<B>1 BEGIN B;
<B>0 BEGIN INITIAL;
<B>[^01\n] BEGIN DEAD;
<B>\n BEGIN INITIAL; {printf("Accepted\n");}

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

%%

intmain()
{
   printf("Enter String\n");
   yylex();
return0;
}
```

Output:

```
Enter String
101011
Accepted
0101100
Not Accepted
0101
Not Accepted
asd
Invalid
```

Q12. Design a DFA in LEX Code which accepts string having even number of 'a' over input alphabet {a,b}.

Code:

```
%{
%}

%s A DEAD

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

<A>a BEGIN INITIAL;
<A>b BEGIN A;
<A>[^ab\n] BEGIN DEAD;
<A>\n BEGIN INITIAL; {printf("Not Accepted\n");}

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

%%

int yywrap()
{
return 1;
}
int main()
{
  printf("Enter String\n");
```

```
  yylex();
return 0;
}
```

Output:

```
Enter String
aab
Accepted
ababa
Not Accepted
bbaaa
Not Accepted
23aa
Invalid
```

Q13. Design a DFA in LEX Code which accepts string containing odd number of 0 and even number of 1.

Code:

```
%{
%}

%s A B C DEAD

%%
<INITIAL>1 BEGIN A;
<INITIAL>0 BEGIN B;
<INITIAL>[^01\n] BEGIN DEAD;
<INITIAL>\n BEGIN INITIAL; {printf("Not Accepted\n");}

<A>1 BEGIN INITIAL;
<A>0 BEGIN C;
<A>[^01\n] BEGIN DEAD;
<A>\n BEGIN INITIAL; {printf("Not Accepted\n");}

<B>1 BEGIN C;
<B>0 BEGIN INITIAL;
<B>[^01\n] BEGIN DEAD;
<B>\n BEGIN INITIAL; {printf("Accepted\n");}

<C>1 BEGIN B;
<C>0 BEGIN A;
<C>[^01\n] BEGIN DEAD;
<C>\n BEGIN INITIAL; {printf("Not Accepted\n");}


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

%%

int main()
{
	printf("Enter String\n");
	yylex();
	return 0;
}
```

Output:

```
Enter String
1010110
Accepted
111000
Not Accepted
1111000
Accepted
jas120
Invalid
```

Q14. Design a DFA in LEX Code to Identify and print Integer & Float Constants and Identifier.

Code:

```
%{
%}

%s A B C DEAD

%%
<INITIAL>[0-9]+ BEGIN A;
<INITIAL>[0-9]+[.][0-9]+ BEGIN B;
<INITIAL>[A-Za-z_][A-Za-z0-9_]* BEGIN C;
<INITIAL>[^\n] BEGIN DEAD;
<INITIAL>\n BEGIN INITIAL; {printf("Not Accepted\n");}

<A>[^\n] BEGIN DEAD;
<A>\n BEGIN INITIAL; {printf("Integer\n");}

<B>[^\n] BEGIN DEAD;
<B>\n BEGIN INITIAL; {printf("Float\n");}

<C>[^\n] BEGIN DEAD;
<C>\n BEGIN INITIAL; {printf("Identifier\n");}


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

%%

intyywrap()
{
 return1;
}

intmain()
{
 printf("Enter String\n");
 yylex();
 return0;
}
```

Output:

```
Enter String
123
Integer
12.54
Float
dsa
Identifier
123dsa
Invalid
```