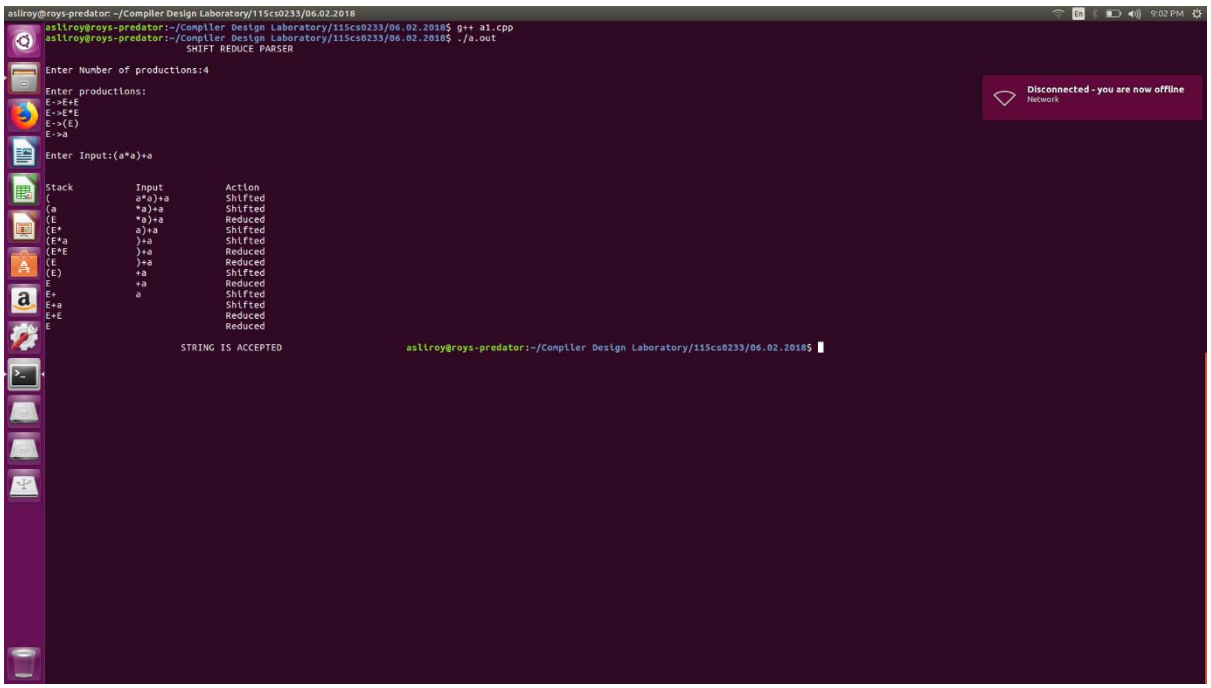
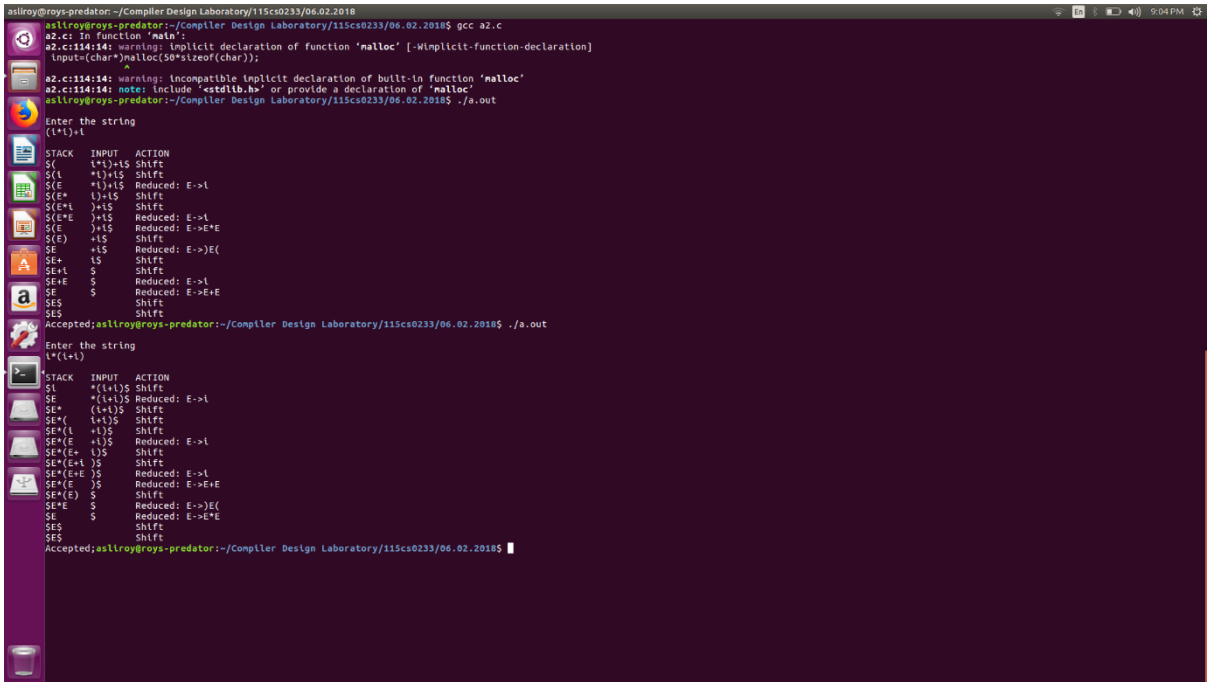
```
    sc=ip[i];
    stack[stpos]=sc;
    i++;stpos++;

  }while(strlen(stack)!=1 && stpos!=lip);

  if(strlen(stack)==1)
  {
    cout<<"\n\n \t\t\tSTRING IS ACCEPTED\t\t\t";
  }
  else
      cout<<"\n\n \t\t\tSTRING IS REJECTED\t\t\t";

  return 0;
}
```

Output:

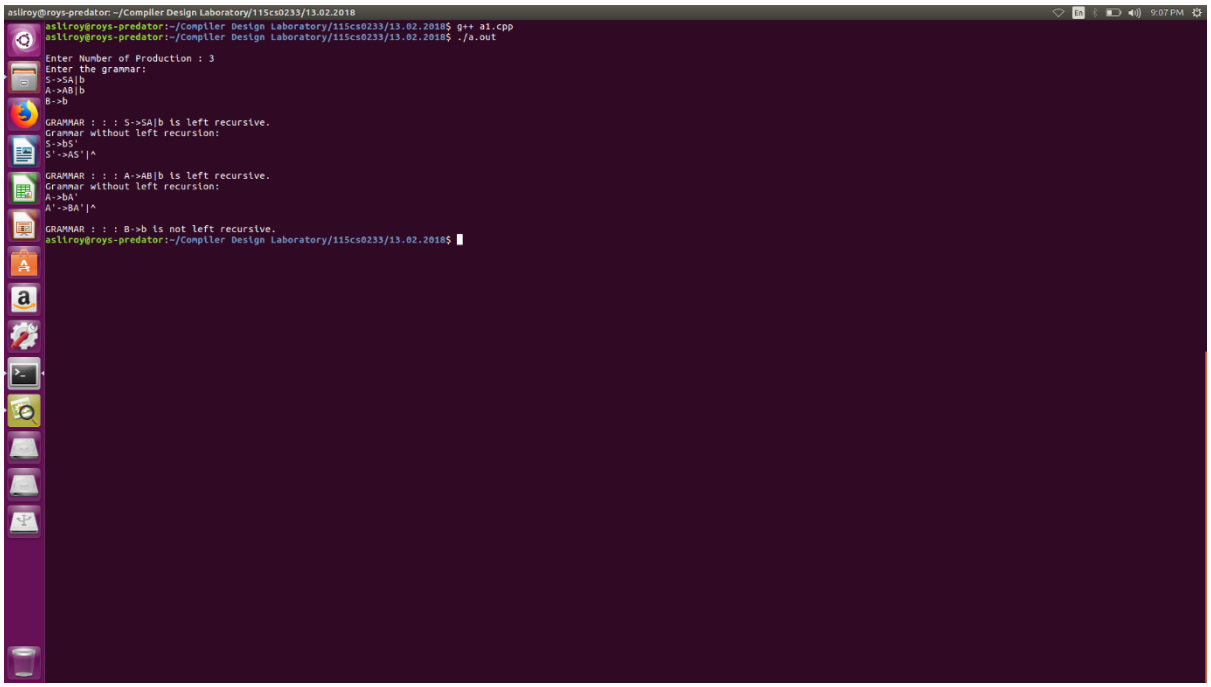


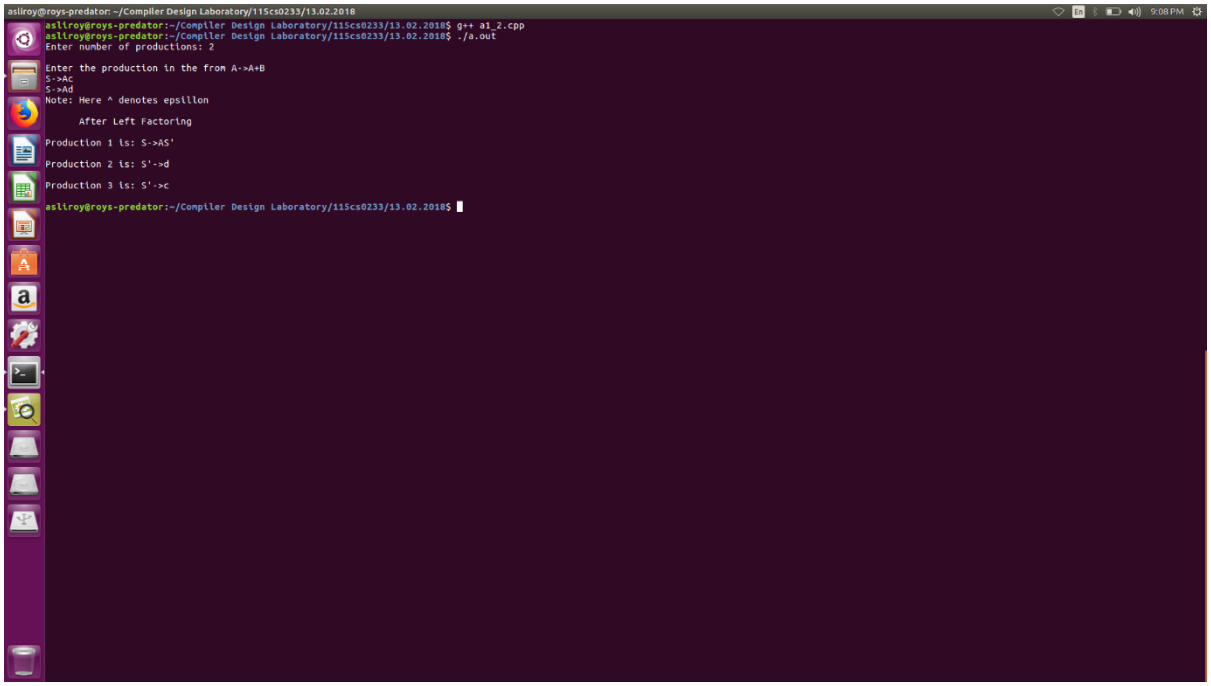
```
else
    printf("\nNot Accepted;");
}
```

Output:



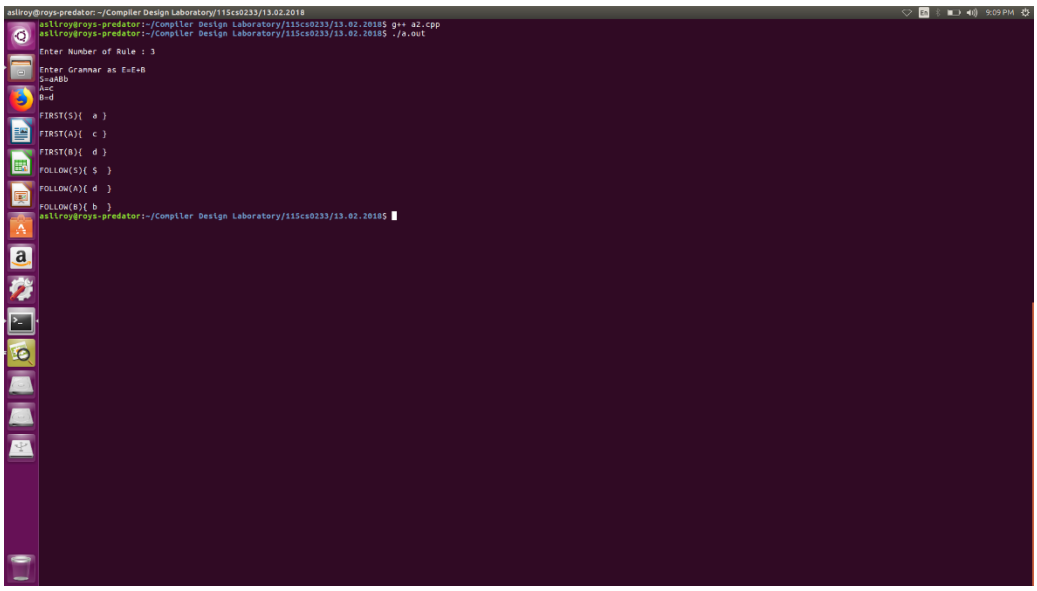
Output:


asliroy@roys-predator: ~/Compiler Design Laboratory/11Scs0233/13.02.2018
9:07 PM
asliroy@roys-predator:~/Compiler Design Laboratory/11Scs0233/13.02.2018$ g++ a1.cpp
asliroy@roys-predator:~/Compiler Design Laboratory/11Scs0233/13.02.2018$ ./a.out
Enter Number of Production : 3
Enter the grammar:
S->SA|b
A->AB|b
B->b
GRAMMAR : : : S->SA|b is left recursive.
Grammar without left recursion:
S->bS'
S'->AS'|^
GRAMMAR : : : A->AB|b is left recursive.
Grammar without left recursion:
A->bA'
A'->BA'|^
GRAMMAR : : : B->b is not left recursive.
asliroy@roys-predator:~/Compiler Design Laboratory/11Scs0233/13.02.2018$


asliroy@roys-predator: ~/Compiler Design Laboratory/11Scs0233/13.02.2018
9:08 PM
asliroy@roys-predator:~/Compiler Design Laboratory/11Scs0233/13.02.2018$ g++ a1_2.cpp
asliroy@roys-predator:~/Compiler Design Laboratory/11Scs0233/13.02.2018$ ./a.out
Enter number of productions: 2
Enter the production in the from A->A+B
S->Ac
S->Ad
Note: Here ^ denotes epsillon
After Left Factoring
Production 1 is: S->AS'
Production 2 is: S'->d
Production 3 is: S'->c
asliroy@roys-predator:~/Compiler Design Laboratory/11Scs0233/13.02.2018$

```
for(i = 0; i < l; i++) {
k = 0;
funcFirst(nonT[i]);
printf("\nFIRST(%c){ ", nonT[i]);
for(j = 0; j < strlen(vFirst); j++)
printf(" %c", vFirst[j]);
printf(" }\n");
}
int s = 0;
for(s = 0; s < l; s++) {
m = 0;
printf("\nFOLLOW(%c){ ", nonT[s]);
funFollow(nonT[s]);
for(i = 0; i < m; i++)
printf("%c ", vFollow[i]);
printf(" }\n");
}
}
```

Output:



ASSIGNMENT-8

1. Design an LR(0) parser that will check the validity of proposition logic expressions and generate its truth table. Also check satisfiablility/tautology/fallacy of the expression.

Valid Tokens

1. A-Z (excluding T and F), a-z (excluding t and f) are tokens of length 1 represent Boolean variables whole values are either T/t for true or F/f for false.

2. T/t and F/f are constants representing true and false respectively.

3. Operator /\ stands for 'AND'.

4. Operator \/ stands for 'OR'.

5. Operator ~ stands for 'NOT'.

6. Operator -> stands for 'implication'.

7. Operator <-> stands for 'if and only if'.

8. Operator (stands for 'opening parenthesis'.

9. Operator) stands for 'closing parenthesis'.

Operator	Precedence	Associativity
highest	()	Left
	~	Right
	<->	Left
	->	Left
	/\	Left
lowest	\/	Left

Errors Handled by this system

1. Lexical Error occurs when invalid tokens are found. The error and its position in the imput string should be displayed.

2. Syntax or Parse Error:

(i) Incomplete expression.

(ii) Operator missing when two operands are consecutive.

(iii) Expression missing when open parenthesis and close parenthesis are consecutive elements in the input string and vice versa.

(iv) Operand missing for a binary operator.

(v) Operand missing for an unary operator.

(vi) Consecutive Binary Operators -> operand missing.

(vii) Missing of an '(' for a ')'.

Code:

```
#include<stdio.h>
#include<string.h>

//Declaration of global variables
char str[50];//user input string,a proposition logic expression
char lex[50];//stores the output of lex_ana
char token[50];
char postfix[50];//holds the postfix expression
char stack[50];//stack used for parsing,postfix generation and evaluation
int lex_ana();//token generation
int syn_ana();//syntax analysis with the help of an LR(0) parser.
unsigned long int mask[130];//masks out the bit position of a variable
char var[20];//holds the variable names
int postfix_gen();//generates postfix
void postfix_epr(int n);//evaluates the intermediate expressions
void truth_table(int n);//generates truth table for a given expression
int lex_ana()//Function for lexical analysis of the w.f.f
{
   char*p,*q,*t;
          int i,j,flag=1;
          p=str;
```

```
   q=lex;
   t=token;
   for(i=1,j=1;*p;i++)
     {

if(((*p>='A')&&(*p<='Z'))||((*p>='a')&&(*p<='z')))
    *q++=*p++,*t++='i';
                else
if((*p==')')||(*p=='(')||(*p=='~'))
                                *t++=*q++=*p++;


                else
 if((*p=='/')&&(*(p+1)=='\\'))
                            *t++=*q++='.',p+=2;
                 else
 if((*p=='\\')&&(*(p+1)=='/'))
                            *t++=*q++='+',p+=2;
                 else
 if((*p=='-')&&(*(p+1)=='>'))
                            *t++=*q++='*',p+=2;
                 else
 if((*p=='<')&&(*(p+1)=='-')&&(*(p+2)=='>'))
                             *t++=*q++='/',p+=3;
 else
                 {
                             printf("\nLexical Error:%d.At position %d,%c is an invalid token.",j,i,*p);
    flag=0;
                                    p++;j++;
 }


     }
```

```
                        *t++=*q++='$';
                        *t=*q='\0';
return flag;
}
int syn_ana()
{
 int j,prev;
 char*p,*top;
 char sym;
 int pos=1,error=0,t=0,state=0,accept=0;
  p=token;top=stack;*top=0;prev=0;
 for(sym=*p,state=0,j=0;!accept;j++)
 {
            /*if(j==10)
                        break;*/
           //printf("state %d  symbol %c   string sym %c\n",state,sym,*p);
             switch(state)
             {
               case 0:

                                switch(sym)
                                {
                                 case 'E':*(++top)=1;t++;state=1;sym=*p;break;
          case 'T':*(++top)=2;t++;state=2;sym=*p;break;
          case '(':*(++top)='(';*(++top)=4;pos++;state=4;sym=*(++p);break;
          case 'i':*(++top)='i';*(++top)=5;pos++;state=5;sym=*(++p);break;
          case '~':*(++top)='~';*(++top)=3;pos++;state=3;sym=*(++p);break;
         /*error handler*/
                                 case ')':printf("Parse Error %d at position %d No ( for
).\n",++error,pos++);break;
          case '$':printf("Parse Error %d at position %d Incomplete
expression.\n",++error,pos++);break;
```

```
        case '+' :printf("Parse Error %d at position %d Left side operand of binary operator /\\ not found.\n",++error,pos++);break;

        case '.' :printf("Parse Error %d at position %d Left side operand of binary operator \\/ not found.\n",++error,pos++);break;

        case '*' :printf("Parse Error %d at position %d Left side operand of binary operator -> not found.\n",++error,pos++);break;

        case '/' :printf("Parse Error %d at position %d Left side operand of binary operator <-> not found.\n",++error,pos++);

                                }

    break;

                case 1:

                                switch(sym)

                                {

        case '$':*(++top)='$';*(++top)=6;pos++;state=6;sym=*(++p);break;

        case '.':*(++top)='.';*(++top)=7;pos++;state=7;sym=*(++p);break;

        case '+':*(++top)='+';*(++top)=8;pos++;state=8;sym=*(++p);break;

        case '*':*(++top)='*';*(++top)=9;pos++;state=9;sym=*(++p);break;

        case '/':*(++top)='/';*(++top)=10;pos++;state=10;sym=*(++p);break;

                                /*error handler*/

        case ')':printf("Parse Error %d at position %d ( missing for ).\n",++error,pos++);break;

                                default :printf("Parse Error %d at position %d Binary operator missing.\n",++error,pos++);

                                }

    break;

                                case 2://reduce by E->T

                                        *(--top)='E';state=*(top-1);t--;sym='E';

                break;

                                case 3:

                  switch(sym)

                                 {

            case 'T':*(++top)=11;t++;state=11;sym=*p;break;

            case '(':*(++top)='(';*(++top)=4;pos++;state=4;sym=*(++p);break;

            case 'i':*(++top)='i';*(++top)=5;pos++;state=5;sym=*(++p);break;
```

```
        case '~':*(++top)='~';*(++top)=3;pos++;state=3;sym=*(++p);break;

                        /*error handler*/

                                case '$':printf("Parse Error %d at position %d Incomplete
expression.\n",++error,pos++);break;

                        default :printf("Parse Error %d at position %d Operand of Unary operator
~ missing.\n",++error,pos++);

    }

            break;

                    case 4:

                            switch(sym)

                                    {

        case '(':*(++top)='(';*(++top)=4;pos++;state=4;sym=*(++p);break;

        case '~':*(++top)='~';*(++top)=3;pos++;state=3;sym=*(++p);break;

        case 'i':*(++top)='i';*(++top)=5;pos++;state=5;sym=*(++p);break;

                        case 'E':*(++top)=12;t++;state=12;sym=*p;break;

        case 'T':*(++top)=2;t++;state=2;sym=*p;break;

                        /*error handler*/

                                    case '+':printf("Parse Error %d at position %d Left operand
of \\/ missing.\n",++error,pos++);break;

                                    case '.':printf("Parse Error %d at position %d Left operand
of /\\ missing.\n",++error,pos++);break;

                                    case '*':printf("Parse Error %d at position %d Left operand
of ->  missing.\n",++error,pos++);break;

                                    case '/':printf("Parse Error %d at position %d Left operand
of <-> missing.\n",++error,pos++);break;

                                    case '$':printf("Parse Error %d at position %d Incomplete
expression.\n",++error,pos++);break;

                                    case ')':printf("Parse Error %d at position %d No expression
between ().\n",++error,pos++);

                                    }

  break;

                        case 5://reduce by T->i

                                *(--top)='T';state=*(top-1);t--;sym='T';

            break;
```

```
                        case 6://accept
                                accept=1;if(!error)printf("The given expression is a valid w.f.f.\n");
                    break;
            case 7: case 8: case 9: case 10:
                switch(sym)
                                        {
                                         case 'T':*(++top)=13;state=13;sym=*p;break;
                 case '(':*(++top)='(';*(++top)=4;pos++;state=4;sym=*(++p);break;
                 case 'i':*(++top)='i';*(++top)=5;pos++;state=5;sym=*(++p);break;
                 case '~':*(++top)='~';*(++top)=3;pos++;state=3;sym=*(++p);break;//NEW
                  /*error handler*/
                 case ')':printf("Parse Error %d at position %d Right side operand of binary operator missing.\n",++error,pos++);break;
                 case '$':printf("Parse Error %d at position %d Incomplete expression.\n",++error,pos++);break;
                 default :printf("Parse Error %d at position %d Two consecutive binary operators.\n",++error,pos++);
                                        }
          break;
                        case 11://reduce by T->~T
                                top-=3;t-=3;*top='T',state=*(top-1);sym='T';
                    break;
                        case 12:
                                switch(sym)
                                {
                 case '.':*(++top)='.';*(++top)=7;t+=2;state=7;sym=*(++p);break;
                 case '+':*(++top)='+';*(++top)=8;pos++;state=8;sym=*(++p);break;
                 case '*':*(++top)='*';*(++top)=9;pos++;state=9;sym=*(++p);break;
                 case '/':*(++top)='/';*(++top)=10;pos++;state=10;sym=*(++p);break;
                 case ')':*(++top)=')';*(++top)=14;pos++;state=14;sym=*(++p);break;
                                /*error handler*/
                                 case '~': case '(':
```

```
                                                printf("Parse Error %d at position %d Binary operator
missing.\n",++error,pos++);break;
                                        case 'i':printf("Parse Error %d at position %d Binary operator or )
missing.\n",++error,pos++);break;
            case '$':printf("Parse Error %d at position %d Incomplete expression.\n",++error,pos++);
            }
                    break;
                            case 13://reduce by E->E/\T,E->E\/T,E->E->T,E->E<->T
                                    top-=5;t-=5;*top='E',state=*(top-1);sym='E';
                    break;
          case 14://reduce by T->(E)
                                    top-=5;t-=5;*top='T',state=*(top-1);sym='T';break;
                                }
  if((!(*p))&&(error)) break;
  else
  if(prev<error){state=0;top=stack;prev=error;sym=*(++p);}
 }
 return error;
}
int postfix_gen()
{
 char*p=lex;
 int end=0,i,k=0;
 unsigned long int t;
 char*top=stack;
 char*r=postfix;
 *top='$';
 for(;!end;p++)
 {
   switch(*p)
           {
           case '$':
```

```
                    end=1;
                    while(*top!='$')
                      *r++=*top--;
 break;
case ')':
                    while(*top!='(')
                              *r++=*top--;
                     top--;
 break;
        case '(':
                    *(++top)='(';
         break;
        case '~':
                    *(++top)='~';
 break;
        case '/':
                    switch(*top)
                    {
                    case '~':;case '/':;
                    *r++=*top;
                    *top='/';
                    break;
                    default:
   *(++top)='/';
   }
break;
                    case '*':
        switch(*top)
                    {
           case '~':;case '/':;case '*':;
                    *r++=*top;
```

```
                *top='*';
                break;
                default:
  *(++top)='*';
  }
break;
      case '.':
              switch(*top)
              {
              case '~':;case '/':;case '*':;case '.':;
               *r++=*top;
               *top='.';
               break;
               default:
  *(++top)='.';
  }
 break;
              case '+':
              switch(*top)
              {
              case '~':;case '/':;case '*':;case '.':;case '+':;
               *r++=*top;
               *top='+';
               break;
               default:
  *(++top)='+';
  }
              break;
              default :
                      *r++=*p;
                      for(i=0,t=1;i<k;i++,t<<=1)
```

```
                                        if(var[i]==*p)
                                                break;
                                        if(i==k)
                                        {
                                                var[i]=*p;
                                                mask[*p]=t;
                    k++;
                                        }
            }
 }
 *r++='$';
 *r='\0';
  return k;
}
void postfix_epr(int n)
{
  int end=0,op=1,t;
   char*p,*top;
printf("TRUTH TABLE OF THE GIVEN EXPRESSION\n");
 top=stack;
 *top='$';
 p=postfix;
 for(t=0;t<n;t++)
            printf("%c ",var[t]);
 printf("  ");
 for(;!end;p++)
 {

            switch(*p)
            {
            case '~':
```

```
                printf("T%d=~",op);
                if(((*top>='A')&&(*top<='Z'))||((*top>='a')&&(*top<='z')))
                   printf("%c",*top);
                else
   printf("T%d",*top);
                *top=op++;
                printf("   ");
break;
        case '/':;case '*':;case '.':;case '+':;
                printf("T%d=",op);
 for(t=1;t>=0;t--)
        {
                if(((*(top-t)>='A')&&(*(top-t)<='Z'))||((*(top-t)>='a')&&(*(top-t)<='z')))
                  printf("%c",*(top-t));
                else
                        printf("T%d",*(top-t));
                if(t)
                {
                 switch(*p)
                 {
    case '/':printf("<->");break;
                  case '*':printf("->");break;
                  case '.':printf("/\\");break;
                  case '+':printf("\\/");
                 }
  }
        }
        top-=2;
        *(++top)=op++;
        printf("   ");
        break;
```

```
            case '$':
                        printf("Result=T%d",*top);
                        end=1;
    break;
            default:
                        *(++top)=*p;
    }
 }
 printf("\n");
}
void truth_table(int n)
{

 int i,j,end,pow=1,t=0;
 char*p,*top;
 for(i=n;i>0;i--)
                pow*=2;
 for(i=pow-1;i>=0;i--)
  {
    printf("\n");
    for(j=0;j<n;j++)
                        printf(((unsigned)mask[var[j]]&(unsigned)i)?"T ":"F ");
             printf("       ");
            end=0;
    top=stack;
    p=postfix;
 for(;!end;p++)
  {

            switch(*p)
            {
```

```
        case '~'://complement
        *top=!(*top);
        printf((*top)?"T ":"F ");
        printf("      ");
break;
        case '/'://<->
        *(top-1)=!(((unsigned)*(top-1))^((unsigned)*top));
        top--;
        printf((*top)?"T ":"F ");
        printf("      ");
break;
        case '*'://->
        *(top-1)=!((unsigned)*(top-1))|((unsigned)*top);
        top--;
        printf((*top)?"T ":"F ");
        printf("      ");
break;
        case '.': //  ^
        *(top-1)=((unsigned)*(top-1))&((unsigned)*top);
        top--;
        printf((*top)?"T ":"F ");
                printf("      ");

 break;
        case '+':;// \/
        *(top-1)=((unsigned)*(top-1))|((unsigned)*top);
        top--;
        printf((*top)?"T ":"F ");
        printf("      ");
break;
        case '$':
```

```
            end=1;if(*top) t++;
    break;
            default:
                    *(++top)=((unsigned)mask[*p])&((unsigned)i)?1:0;
           }
  }
 }
 printf("\nThe given w.f.f. is ");
 printf((t)?(t==pow)?"Tautology":"Satisfiable":"Unsatisfiable");
}
int main()
{
 int i,n;
 printf("Enter an logic expression(w.f.f.):");
 scanf("%s",str);
 //strcpy(str,"((a/\\b\\/c)->(d/\\~e))<->(~f->(g/\\h)/\\~i<->j/\\k\\/l)");
 printf("Evaluating string: %s\n",str);
 for(i=1;i<=1;i++)
 {
  if(lex_ana())
  {
           if(!syn_ana())
            {
      n=postfix_gen();
              postfix_epr(n);
              truth_table(n);
      printf("\n");
           }
           else
                   printf("The given expression is not a w.f.f.");
  }
```

```
   printf("\n");
  }
 return 0;
}
```

Output: