1.USE LEX TOOL TOOLTO IMPLEMENT LEXICAL ANALYZER

```
%option noyywrap
%{
        #include<stdio.h>
        void yyerror(char *);
%}
letter [a-z A-Z]
digit [0-9]
op [-+*]
%%
else|int|float {printf("%s is a keyword",yytext);}
{digit}+ {printf("%s is a number",yytext);}
{letter}({letter}|{digit})* {printf("%s is a identifier",yytext);}
{op}+  {printf("%s is a operator",yytext);}
. yyerror("error");
%%
void yyerror(char *s)
{
fprintf(stderr,"%s\n",s);
}
int main()
{
        yylex();
        return 0;
}
```

2.LEX TOOL TOOLTO IMPLEMENT PARSER USING AMBIGUOUS

LEX FILE:

```
%option noyywrap
%{
        #include<stdio.h>
        #include"y.tab.h"
        void yyerror(char *s);
        extern int yylval;
%}
digit [0-9]
%%
{digit}+  {yylval=atoi(yytext);return NUM;}
[-+*/\n]  {return *yytext;}
\(        {return *yytext;}
\)        {return *yytext;}
.         {yyerror("syntax error");}
%%
```

YACC FILE:

```
%{
          #include<stdio.h>
          void yyerror(char*);
          extern int yylex(void);
%}
%token NUM
%%
S:
S E '\n'        {printf("%d\n",$2);}
|
;
E:
E '+' E         {$$=$1+$3;}
|E '-' E        {$$=$1-$3;}
|E '*' E        {$$=$1*$3;}
|E '/' E        {$$=$1/$3;}
```

```
|'(' E ')'      {$$=$2;}
|NUM            {$$=$1;}
%%
void yyerror(char *S)
{
printf("%S",S);
}
int main()
{
yyparse();
return 0;
}
```

3.LEX AND YACC TOLL TO IMPLEMENT UMAMIGOUS GRAMMER

LEX FILE:

```
%option noyywrap
%{
        #include<stdio.h>
        #include"y.tab.h"
        void yyerror(char *s);
        extern int yylval;
%}
digit [0-9]
%%
{digit}+  {yylval=atoi(yytext);return NUM;}
[-+*/\n]  {return *yytext;}
\(        {return *yytext;}
\)        {return *yytext;}
.         {yyerror("syntax error");}
%%
```

YACC FILE:

```
%{
          #include<stdio.h>
          void yyerror(char *);
          extern int yylex(void);
%}
%token NUM
%%
S:
S E '\n'        {printf("%d\n",$2);}
|
;
E:
E '+' T         {$$=$1+$3;}
|E '-' T        {$$=$1-$3;}
|T              {$$=$1;}
T:
T '*' F         {$$=$1*$3;}
|T '/' F        {$$=$1/$3;}
|F              {$$=$1;}
F:
|'(' E ')'      {$$=$2;}
|NUM            {$$=$1;}
%%
void yyerror(char *S)
{
printf("%S",S);
}
int main()
{
yyparse();
return 0;
}
```

4.LEX TOOL TO IMPLEMENT CALCULATOR

```
%{
       int a,b,flag=0;
%}
dig [0-9]*
add "+"
sub "-"
mul "*"
div "/"
%%
{dig} {dig();}
{add} {flag=1;}
{sub} {flag=2;}
{mul} {flag=3;}
{div} {flag=4;}
\n {printf("The answer is:%d\n",a);}
%%
dig()
{
    if(flag==0)
    {
       a=atof(yytext);
    }
    else
    {
          b=atof(yytext);
          switch(flag)
          {
              case 1:
                    a=a+b;
                    break;
              case 2:
                    a=a-b;
                    break;
              case 3:
                    a=a*b;
                    break;
              case 4:
                    a=a/b;
                    break;
          }
     }
}
int main()
{
      yylex();
      return 0;
}
int yywrap(void) {}
```

5.IMPLEMENT ERECURSIVE DECENT PARSER ALGORITHM

```
#include<stdio.h>
#include<stdlib.h>
char |;

void match(char c)
{
     if(i==c)
        i=getchar();
     else
     {
        printf("Invalid Input\n");
        exit(0);
     }
}

void B()
{
  if(i=='b')
  {
     match('b');
  }
  else
  {
     printf("Invalid Input\n");
     exit(0);
  }
}
void A()
{
  if(i=='a')
  {
     match('a');
     B();
  }
  else
  return;
}

void S()
{
 A();
 A();
}

void main()
{
   char input[10];
   printf("Enter String with $ at the end\n");
   i=getchar();
   S();
   if(i=='$')
   {
       printf("\nParsing Successful\n");
   }
   else
   {
       printf("Invalid Input\n");
   }
}
```

6.IMPLEMENT SHIFT REDUCE PARSER USING C PROGRAME

```
#include <stdio.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();
int main()
{
        puts("GRAMMAR is E->E+E \n E->E*E \nE->(E) \n E->id");
        puts("enter input string");
        scanf("%s",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{j+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();
                }
        }
}
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+2]='\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;
                   }
}
```

7. IMPLEMENT OPERATOR PRECENDENCE PASER ALGORITNM

```
#include<stdio.h>
#include<conio.h>
void main()
```

```
{
        char stack[20],ip[20],opt[10][10][1],ter[10];
        int i,j,k,n,top=0,col,row;
        clrscr();
        for(i=0;i<3;i++)
         {
           stack[i]=NULL;
           ip[i]=NULL;
           for(j=0;j<3;j++)
           {
             opt[i][j][0]=NULL;
           }
         }
         printf("Enter the no.of terminals:");
         scanf("%d",&n);
         printf("\nEnter the terminals:");
         scanf(" %s",ter);
         printf("\nEnter the table values:\n");
         for(i=0;i<n;i++)
         {
         for(j=0;j<n;j++)
          {
           printf("Enter the value for %c %c:",ter[i],ter[j]);
           scanf(" %s",opt[i][j]);
          }
         }
printf("\nOPERATOR PRECEDENCE TABLE:\n");
for(i=0;i<n;i++){printf("\t%c",ter[i]);}
printf("\n");
for(i=0;i<n;i++)
{
    printf("\n%c",ter[i]);
    for(j=0;j<n;j++)
    {
        printf("\t%c",opt[i][j][0]);

     }
  }
  stack[top]='$';
  printf("\nEnter the input string:");
  scanf(" %s",ip);
  i=0;
  printf("\nSTACK\t\t\tINPUT STRING\t\t\tACTION\n");
  printf("\n%s\t\t\t%s\t\t\t",stack,ip);
  while(i<=strlen(ip))
 {
        for(k=0;k<n;k++)
        {
          if(stack[top]==ter[k])
          row=k;
          if(ip[i]==ter[k])
          col=k;
        }
        if((stack[top]=='$')&&(ip[i]=='$'))
        {
        printf("String is accepted");
        break;
        }
        else if((opt[row][col][0]=='<') ||(opt[row][col][0]=='='))
        {
          stack[++top]=opt[row][col][0];
          stack[++top]=ip[i];
          printf("Shift %c",ip[i]);
          i++;
        }
         else
         {
          if(opt[row][col][0]=='>')
          {
            while(stack[top]!='<')
            --top;
            top=top-1;
            printf("Reduce");
          }
         else
          {
           printf("\nString is not accepted");
           break;
          }
        }
 printf("\n");
 for(k=0;k<=top;k++)
 printf("%c",stack[k]);
 printf("\t\t\t");
 for(k=i;k<strlen(ip);k++)
 printf("%c",ip[k]);
 printf("\t\t\t");
}
getch();
}
```

8.IMPLEMENT THE FRONT END OF THE COMPILER TO PRODUCE THREE ADDRESS CODE

```
#include<stdio.h>
#include<string.h>
void pm();
void plus();
void div();
int i,ch,j,l,addr=100;
char ex[10],exp[10] ,exp1[10],exp2[10],id1[5],op[5],id2[5];
void main()
{
clrscr();
printf("\nEnter the expression with arithmetic operator:");
scanf("%s",ex);
strcpy(exp,ex);
l=strlen(exp);
exp1[0]='\0';

for(i=0;i<l;i++)
{
if(exp[i]=='+'||exp[i]=='-')
{
if(exp[i+2]=='/'||exp[i+2]=='*')
{
pm();
break;
}
else
{
plus();
break;
}
}
else if(exp[i]=='/'||exp[i]=='*')
{
div();
break;
}
}

}
void pm()
```

```
{
strrev(exp);
j=l-i-1;
strncat(exp1,exp,j);
strrev(exp1);
printf("Three address code:\ntemp=%s\ntemp1=%c%ctemp\n",exp1,exp[j+1],exp[j]);
}
void div()
{
strncat(exp1,exp,i+2);
printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n",exp1,exp[i+2],exp[i+3]);
}
void plus()
{
strncat(exp1,exp,i+2);
printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n",exp1,exp[i+2],exp[i+3]);
}
```

9.IMPLEMENT SYMBOL TABLE MANAGEMENT

```
#include<stdio.h>
#include<ctype.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
void main()
{
 int i=0,j=0,x=0,n;
 void *p,*add[5];
 char ch,srch,b[15],d[15],c;
 printf("Expression terminated by $:");
 while((c=getchar())!='$')
 {
  b[i]=c;
  i++;
 }
 n=i-1;
 printf("Given Expression:");
 i=0;
 while(i<=n)
 {
  printf("%c",b[i]);
  i++;
 }
 printf("\n Symbol Table\n");
 printf("\nSymbol \t addr \t type");
 while(j<=n)
 {
  c=b[j];
  if(isalpha(toascii(c)))
  {
   p=malloc(c);
   add[x]=p;
   d[x]=c;
   printf("\n%c \t %d \t identifier\n",c,p);
   x++;
   j++;
  }
  else
  {
   ch=c;
   if(ch=='+'||ch=='-'||ch=='*'||ch=='=')
   {
    p=malloc(ch);
    add[x]=p;
    d[x]=ch;
    printf("\n %c \t %d \t operator\n",ch,p);
    x++;
    j++;
   }}}}
```

10.IMPLEMENTATION OF SIMPLE CODE OPTIMIZATION TECHNIQUE

```
#include<stdio.h>
#include<conio.h>
void main()
{
unsigned int n;
int x;
char ch;
clrscr();
printf("\nEnter N\n");
scanf("%u",&n);
printf("\n1. Loop Roll\n2. Loop UnRoll\n");
printf("\nEnter ur choice\n");
scanf(" %c",&ch);
switch(ch)
{
case '1':
  x=countbit1(n);
  printf("\nLoop Roll: Count of  1's    :  %d" ,x);
  break;
case '2':
  x=countbit2(n);
  printf("\nLoop UnRoll:  Count of 1's  :  %d" ,x);
  break;
default:
  printf("\n Wrong Choice\n");

}
getch();
}
int countbit1(unsigned int n)
{
    int bits = 0,i=0;
    while (n != 0)
    {
 if (n & 1) bits++;
 n >>= 1;
 i++;
    }
    printf("\n no of iterations  %d",i);
    return bits;
}
int countbit2(unsigned int n)
{
    int bits = 0,i=0;
    while (n != 0)
    {
 if (n & 1) bits++;
 if (n & 2) bits++;
 if (n & 4) bits++;
 if (n & 8) bits++;
 n >>= 4;
 i++;
    }
    printf("\n no of iterations  %d",i);
    return bits;
}
```

11.CONSTRUCT A SIMPLE CODE GENERATOR

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

```
#include<string.h>
void main()
{
char icode[10][30], str[20], opr[10];
int i=0;
printf("\nEnter the set of intermediate code (terminated by exit):\nâ€□);
do{
        scanf("%s", icode[i]);
}while(strcmp(icode[i++],"exit")!=0);
printf("\nTarget code generation");
printf("\n*******************");
i=0;
do{
        strcpy(str,icode[i]);
        switch(str[3]){
                case '+':
                        strcpy(opr,"ADD");
                        break;
                case '-':
                        strcpy(opr,"SUB");
                        break;
                case '*':
                        strcpy(opr,"MUL");
                        break;
                case '/':
                strcpy(opr,"DIV");
                        break;
        }
        printf("\n\tMov %c,R%d", str[2],i);
        printf("\n\t%s %c,R%d", opr,str[4],i);
        printf("\n\tMov R%d,%c", i,str[0]);
}while(strcmp(icode[++i],"exit")!=0);
 }
```