EXP1 : TOKEN SEPERATION

```
%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 an identifier",yytext);}
{op}+    {printf("%s is an operator",yytext);}
. yyerror("error");
%%
void yyerror(char *s)
{
fprintf(stderr,"%s\n",s);
}
int main()
{
      yylex();
         return 0;
}
```

EXP2: DIGIT OR NOT

```
%option noyywrap
%{
#include<stdio.h>
%}
digit    [0 - 9]
letter   [a-z A-Z]
%%
{digit}+  {printf("%s is digit",yytext);}
{letter}* {printf("%s is identifier",yytext);}
.          ;
%%
int main()
{
        printf("Enter the Input");
        yylex();
        printf("end");
        return 0;
}
```

ODD OR EVEN

```
%option noyywrap
%{
	#include<stdio.h>
	int i;
%}

%%
[0-9]+		{i=atoi(yytext);
	if(i%2==0)
	printf("even no");
	else
	printf("odd no");
	};
%%
int main()
{
	yylex();
	return 0;
}
```

EXP3 :AMBIGOUS GRAMMER

.L

```
%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");}
%%
```

.Y

```
%{
        #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;
}
```

EXP4:UNAMBIGOUS GRAMMER

```
.L
%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");}
%%
.Y
%{
        #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;
}
```

EXP5: DESKTOP CALCULATOR

.L

```
%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;}
[a-z]           {yylval=toascii(*yytext)-97;return ID;}
[A-Z]           {yylval=toascii(*yytext)-65;return ID;}
[-+*/=\n]       {return *yytext;}
\(              {return *yytext;}
\)              {return *yytext;}
.               {yyerror("syntax error");}
%%Desktop Calculator
```

.Y

```
%{
        #include<stdio.h>
        void yyerror(char*);
        extern int yylex(void);
        int val[26];
%}
%token NUM ID
%%
```

```
S:
S E '\n'                {printf("%d\n",$2);}
| S ID '=' E '\n'       {val[$2]=$4;}
|
;
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;}
|ID             {$$=val[$1];}
%%
void yyerror(char *s)
{
printf("%s",s);
}
int main()
{
yyparse();
return 0;
}
```

EXP 6:SHIFT REDUCE PARSER

```
/** SHIFT REDUCE PARSING**/
#include<stdio.h>
#include<conio.h>
#include<string.h>
int z,i,j,c;
char a[16],stk[15];
void reduce();
void main()
  {  clrscr();
     puts("GRAMMAR is E->E+E \n E->E*E \n E->(E) \n E->a");
     puts("enter input string ");
     gets(a);
     c=strlen(a);
     a[c]='$';
     stk[0]='$';
     puts("stack \t input \t action");
     for(i=1,j=0;j<c; i++,j++)
      {
          if(a[j]=='a')
            {
               stk[i]=a[j];
               stk[i+1]='\0';
               a[j]=' ';
               printf("\n%s\t%s\tshift->a",stk,a);
               reduce();
            }
          else
```

```
            {
              stk[i]=a[j];
              stk[i+1]='\0';
              a[j]=' ';
              printf("\n%s\t%s\tshift->%c",stk,a,stk[i]);
              reduce();
            }
      }
            if(a[j]=='$')
            reduce();
            if((stk[1]=='E')&&(stk[2]=='\0'))
            printf("\n%s\t%s\tAccept",stk,a);
            else
            printf("\n%s\t%s\terror",stk,a);
            getch();
   }
void reduce()
   {
       for(z=1; z<=c; z++)
       if(stk[z]=='a')
            {
              stk[z]='E';
              stk[z+1]='\0';
              printf("\n%s\t%s\tReduce by E->a",stk,a);
            }
      for(z=1; 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\tReduce by E->E+E",stk,a);
        i=i-2;
      }
 for(z=1; 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\tReduce by E->E*E",stk,a);
        i=i-2;
      }
  for(z=1; z<=c; z++)
  if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')')
      {
        stk[z]='E';
        stk[z+1]='\0';
        stk[z+2]='\0';
        printf("\n%s\t%s\tReduce by E->(E)",stk,a);
        i=i-2;
      }
}
```

EXP7:RECURSIVE DESCENT PARSER

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int i=0,f=0;
char str[30];
void E();
void Eprime();
void T();
void Tprime();
void F();
void E()
{
   printf("\nE->TE'");
   T();
   Eprime();
}
void Eprime()
{
   if(str[i]=='+')
   {
      printf("\n\E'->+TE'");
      i++;
      T();
      Eprime();
      }
      else if(str[i]==')')
      printf("\nE'->^");
```

```
}
void T()
{
   printf("\nT->FT'");
   F();
   Tprime();

}
void Tprime()
{
   if(str[i]=='*')
   {
      printf("\nT'->*FT'");
      i++;
      F();
      Tprime();
      }
      else if((str[i]==')')||(str[i]=='+'))
      printf("\nT'->^");

}
void F()
{
   if(str[i]=='a')
   {
      printf("\nF->a");
      i++;
```

```
        }
        else if(str[i]=='(')
        {
            printf("\nF->(E)");
            i++;
            E();
            if(str[i]==')')
            i++;
            }
            else
            f=1;
}
void main()
{
   int len;
   clrscr();
   printf("Enter the string: ");
   scanf("%s",str);
   len=strlen(str);
   str[len]='$';
   E();
   if((str[i]=='$')&&(f==0))
   printf("\nStringsucessfully parsed!");
   else
   printf("\nSyntax Error!");
         getch();
}
```

EXP8: OPERATOR PRECEDENCE PARSER

```
#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();

}
```

EXP 9 : 3 ADDRESS CODE

```
#include<stdio.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]);
}
```

EXP 10:SYMBOL TABLE MANAGEMENT

```
#include<stdio.h>
#include<ctype.h>
#include<stdlib.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++;
  }}}}
```

EXP11: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):\n");
     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);
}
```