EX NO 1:

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<string.h>
#include<stdlib.h>
#define NULL 0 int size=0;
void Insert();
 void Display();
void Delete();
int Search(char lab[]);
void Modify();
struct SymbTab
{
char label[10],symbol[10];
int addr;
struct SymbTab *next;
};
struct SymbTab *first,*last;
void main()
{
int op,y;
 char la[10];
clrscr();
do
 {
printf("\n\tSYMBOL TABLE IMPLEMENTATION\n");
printf("\n\t1.INSERT\n\t2.DISPLAY\n\t3.DELETE\n\t4.SEARCH\n\t5.MODIFY\n\t
    6.END\n"); printf("\n\tEnter your option : ");
scanf("%d",&op); switch(op)
{
case 1:
Insert();
break;
case 2:
Display();
 break;
case 3:
Delete();
break;
case 4:
printf("\n\tEnter the label to be searched : ");
scanf("%s",la);
y=Search(la);
printf("\n\tSearch Result:");
if(y==1)
printf("\n\tThe label is present in the symbol table\n");
else
printf("\n\tThe label is not present in the symbol table\n");
break;
case 5:
Modify();
break;
case 6:
exit(0);
}
}
while(op<6);
```

```
getch();
}
void Insert()
{
int n;
char l[10];
printf("\n\tEnter the label : ");
scanf("%s",l);
n=Search(l);
if(n==1)
printf("\n\tThe label exists already in the symbol table\n\tDuplicate can't be
    inserted");
else
{
struct SymbTab *p;
p=malloc(sizeof(struct SymbTab));
strcpy(p->label,l);
printf("\n\tEnter the symbol : ");
scanf("%s",p->symbol);
printf("\n\tEnter the address : ");
scanf("%d",&p->addr);
p->next=NULL; if(size==0)
{
first=p; last=p;
}
else
{
last->next=p;
 last=p;
}
size++;
}
printf("\n\tLabel inserted\n");
}
void Display()
{
int i;
struct SymbTab *p;
p=first;
printf("\n\tLABEL\t\tSYMBOL\t\tADDRESS\n");
for(i=0;i<size;i++)
{
printf("\t%s\t\t%s\t\t%d\n",p->label,p->symbol,p->addr);
p=p->next;
}
}
int Search(char lab[])
{
int i,flag=0;
struct SymbTab *p;
p=first;
for(i=0;i<size;i++)
{
if(strcmp(p->label,lab)==0) flag=1;
p=p->next;
}
return flag;
}
```

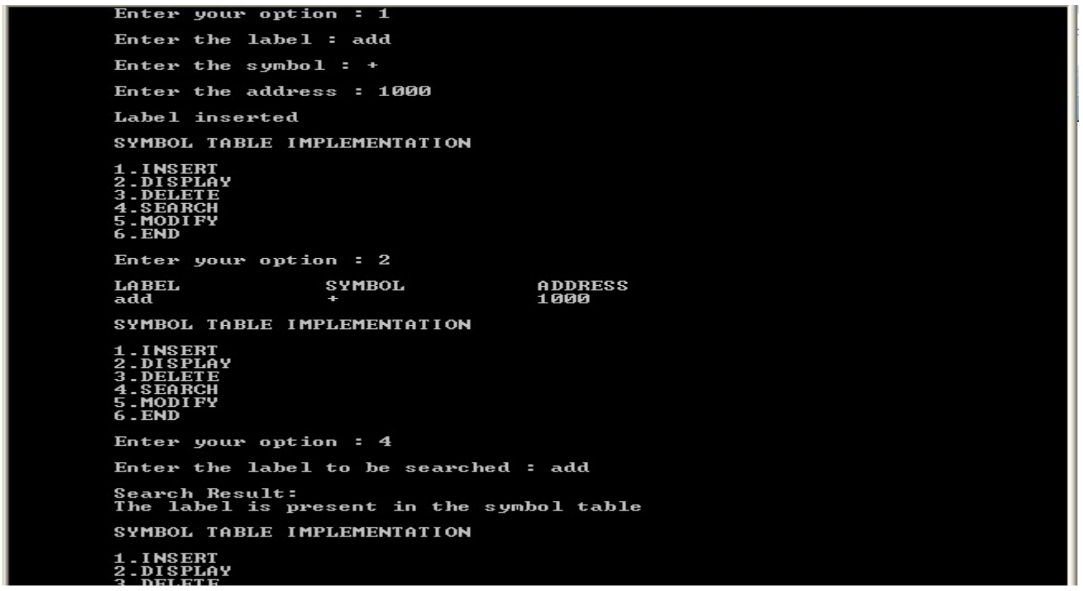
```
void Modify()
{
char l[10],nl[10];
 int add,choice,i,s; struct SymbTab *p; p=first;
printf("\n\tWhat do you want to modify?\n");
printf("\n\t1.Only the label\n\t2.Only the address\n\t3.Both the label and
    address\n"); printf("\tEnter your choice : ");
scanf("%d",&choice); switch(choice)
{
case 1:
printf("\n\tEnter the old label : ");
scanf("%s",l);
s=Search(l);
if(s==0)
printf("\n\tLabel not found\n");
else
{
printf("\n\tEnter the new label : ");
scanf("%s",nl); for(i=0;i<size;i++)
{
if(strcmp(p->label,l)==0) strcpy(p->label,nl);
p=p->next;
}
printf("\n\tAfter Modification:\n");
Display();
}
break; case 2:
printf("\n\tEnter the label where the address is to be modified : ");
scanf("%s",l);
```

```
s=Search(l);
if(s==0)
printf("\n\tLabel not found\n");
else
{
printf("\n\tEnter the new address : ");
scanf("%d",&add);
for(i=0;i<size;i++)
{
if(strcmp(p->label,l)==0) p->addr=add;
p=p->next;
}
printf("\n\tAfter Modification:\n");
Display();
}
 break; case 3:
printf("\n\tEnter the old label : ");
scanf("%s",l);
s=Search(l);
if(s==0)
printf("\n\tLabel not found\n");
else
{
printf("\n\tEnter the new label : ");
scanf("%s",nl);
printf("\n\tEnter the new address : ");
scanf("%d",&add);
for(i=0;i<size;i++)
{
```

```
if(strcmp(p->label,l)==0)
{
strcpy(p->label,nl);
p->addr=add;
}
p=p->next;
}
printf("\n\tAfter Modification:\n");
Display();
}
break;
}
}
void Delete()
{
int a;
char l[10];
struct SymbTab *p,*q;
p=first;
printf("\n\tEnter the label to be deleted : ");
scanf("%s",l);
a=Search(l);
if(a==0)
printf("\n\tLabel not found\n");
else
{
if(strcmp(first->label,l)==0) first=first->next;
else if(strcmp(last->label,l)==0)
{
```

```
q=p->next;
while(strcmp(q->label,l)!=0)
{
 p=p->next; q=q->next;
}
p->next=NULL;
last=p;
}
else
{
q=p->next;
while(strcmp(q->label,l)!=0)
{
p=p->next;
q=q->next;
}
p->next=q->next;
}
size--;
printf("\n\tAfter Deletion:\n");
Display();
}
}
```

OUTPUT:



```
Enter your option : 1
Enter the label : add
Enter the symbol : +
Enter the address : 1000
Label inserted
SYMBOL TABLE IMPLEMENTATION
1.INSERT
2.DISPLAY
3.DELETE
4.SEARCH
5.MODIFY
6.END
Enter your option : 2
LABEL           SYMBOL          ADDRESS
add             +               1000
SYMBOL TABLE IMPLEMENTATION
1.INSERT
2.DISPLAY
3.DELETE
4.SEARCH
5.MODIFY
6.END
Enter your option : 4
Enter the label to be searched : add
Search Result:
The label is present in the symbol table
SYMBOL TABLE IMPLEMENTATION
1.INSERT
2.DISPLAY
```

EX NO 2:

```
#include<stdio.h>
#include<string.h>
main()
{
FILE *fp;
char a[5]={':','-','*','+','='};
char b[8]={'{','}','[',']','(',')'};
char q[20]={'a','b','c','d'};
char p[15][15]={"int","if","void"};
int i,j,k,n,l;
char x,ch,y[7],s[10],z[8],ch1[80],id[60];
printf("********\n Choices are: \n*********");
printf("\n 1. Operators");
printf("\n 2. Special Symbols");
printf("\n 3. Keywords");
printf("\n 4. Identifiers");
printf("\n 5. Exit"); first:;
printf("\n Enter your choice:");
scanf("%d",&n);
switch(n)
{
case 1:
printf("\n 1. Operators");
 for(i=0;i<strlen(a);i++)
{
fp=fopen("in6.txt","r");
do
{
   ch=fgetc(fp);
   if(ch==a[i])
{
printf("\n%c\n",ch);
break;
}
}
while(!feof(fp));
}
fclose(fp);
goto first;
case 2:
printf("\n 2. Special Symbols");
for(j=0;j<strlen(b);j++)
{
fp=fopen("in6.txt","r");
do
{
   x=fgetc(fp);
   if(x==b[j])
{
printf("\n%c\n",b[j]);
break;
}
}
while(x!=EOF);
}
fclose(fp);
goto first;
```

```
break;
case 3:
printf("\n 3. Keywords");
fp=fopen("in6.txt","r");
l=0;
x=getc(fp);
while(x!=EOF&&x!='(')
{
id[l]=x;
l++;
x=getc(fp);
 }
id[l]='\0';
fclose(fp);
printf("\n%s\n",id);
goto first;
break;
case 4:
printf("\n 4. Identifiers");
for(i=0;i<strlen(q);i++)
{
fp=fopen("in6.txt","r");
do
{
ch=fgetc(fp);
if(ch==q[i])
{
printf("\n%c\n",ch);
break;
}
}
while(!feof(fp));
}
fclose(fp);
goto first;
break;
case 5:
printf("5. You Want To Quit Give Y:");
scanf("\n%c\n",&y);
if(getchar()=='y')
exit(0);
else goto first;
break;
}
}
```

OUTPUT:

```
1.      Operators
2.      Special Symbols
3.      Keywords
4.      Identifiers
5.      Exit

Enter your choice:1
1.      Operators
+
=

Enter your choice: 2
2.      Special Symbols (
)
Enter your choice:3
3.      Keywords if

Enter your choice:4
4.      Identifiers a
b

Enter your choice:5

5.      You Want To Quit Give Y: y
```

EX NO: 3

Lex.l

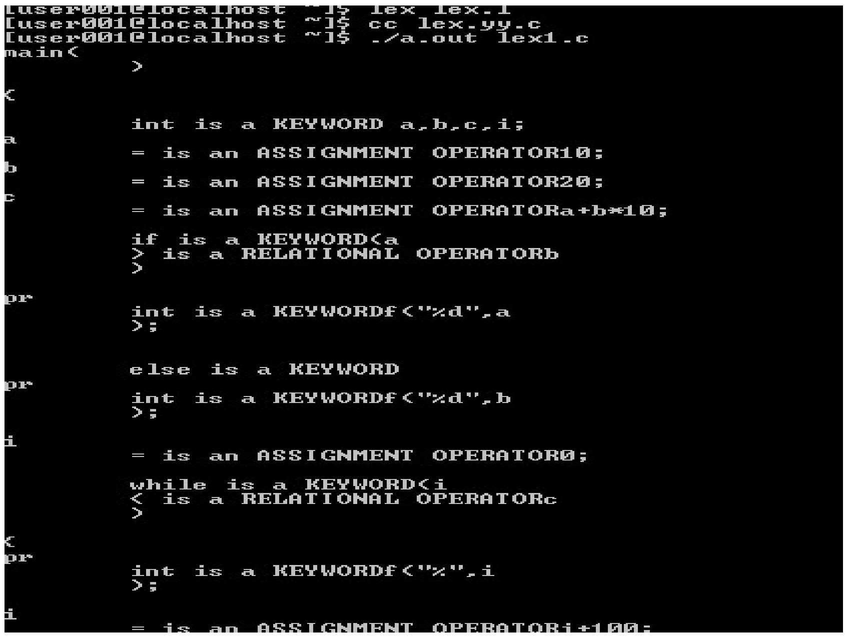
```
%{
int COMMENT=0;
%}
identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* {printf("\n%s is a preprocessor directive",yytext);}
int |
float |
char |
double |
while |
for |
struct |
typedef |
do |
if |
break |
continue |
void |
switch |
return |
else |
goto {printf("\n\t%s is a keyword",yytext);}
"/*" {COMMENT=1;}{printf("\n\t %s is a COMMENT",yytext);}
```

```
{identifier}\( {if(!COMMENT)printf("\nFUNCTION \n\t%s",yytext);}
\{ {if(!COMMENT)printf("\n BLOCK BEGINS");}
\} {if(!COMMENT)printf("BLOCK ENDS ");}
{identifier}(\[[0-9]*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}
\".*\" {if(!COMMENT)printf("\n\t %s is a STRING",yytext);}
[0-9]+ {if(!COMMENT) printf("\n %s is a NUMBER ",yytext);}
\)(\:)? {if(!COMMENT)printf("\n\t");ECHO;printf("\n");}
\( ECHO;
= {if(!COMMENT)printf("\n\t %s is an ASSIGNMENT OPERATOR",yytext);}
\<= |
\>= |
\< |
== |
\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}
%%
int main(int argc, char **argv)
{
FILE *file;
file=fopen("input.c.txt","r");
if(!file)
{
printf("could not open the file");
exit(0);
}
yyin=file;
yylex();
printf("\n");
return(0);
}
int yywrap()
{
return(1);
}
```

Input file.c

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int a,b;
 char d;
 a=8;
 b=7;
}
```

OUTPUT:



```
[user001@localhost ~]$ lex lex.l
[user001@localhost ~]$ cc lex.yy.c
[user001@localhost ~]$ ./a.out lex1.c
main(
        )

{

        int is a KEYWORD a,b,c,i;
a
        = is an ASSIGNMENT OPERATOR10;
b
        = is an ASSIGNMENT OPERATOR20;
c
        = is an ASSIGNMENT OPERATORa+b*10;

        if is a KEYWORD(a
        > is a RELATIONAL OPERATORb
        )

pr
        int is a KEYWORDf("%d",a
        );

        else is a KEYWORD
pr
        int is a KEYWORDf("%d",b
        );

i
        = is an ASSIGNMENT OPERATOR0;

        while is a KEYWORD(i
        < is a RELATIONAL OPERATORc
        )

{
pr
        int is a KEYWORDf("%",i
        );

i
        = is an ASSIGNMENT OPERATORi+100;
```

EX NO: 4

LEX PART:

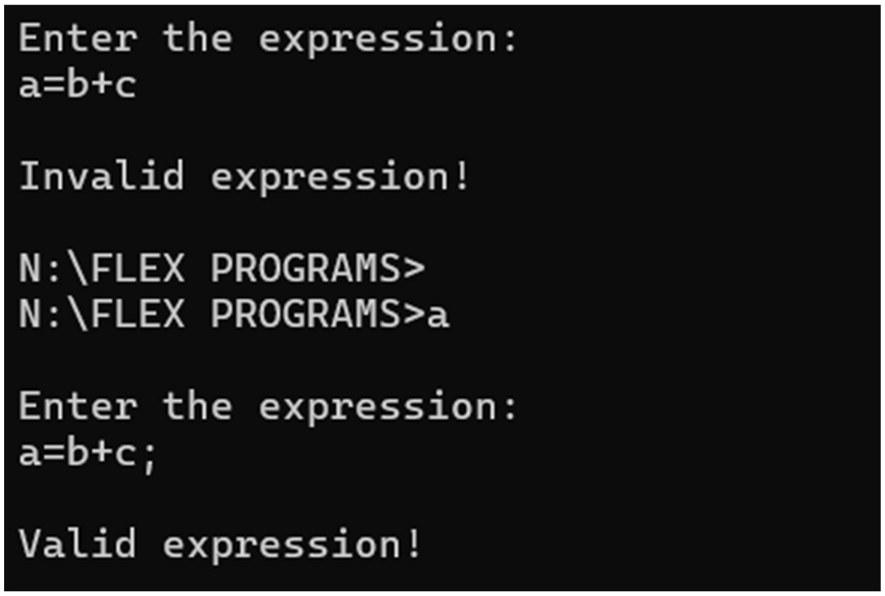
```
%{
   #include "y.tab.h"
%}
%%
[a-zA-Z_][a-zA-Z_0-9]* return id;
[0-9]+(\.[0-9]*)?     return num;
[+/*]                 return op;
.                     return yytext[0];
\n                    return 0;
%%
int yywrap()
{
return 1;
}
```

YACC PART:

```
%{
   #include<stdio.h>
   int valid=1;
%}
%token num id op
%%
start : id '=' s ';'
s :    id x
     | num x
     | '-' num x
     | '(' s ')' x
     ;
x :    op s
     | '-' s
     |
     ;
%%
int yyerror()
```

```
{
   valid=0;
   printf("\nInvalid expression!\n");
   return 0;
}
int main()
{
   printf("\nEnter the expression:\n");
   yyparse();
   if(valid)
   {
      printf("\nValid expression!\n");
   }
}
```

OUTPUT:



```
Enter the expression:
a=b+c

Invalid expression!

N:\FLEX PROGRAMS>
N:\FLEX PROGRAMS>a

Enter the expression:
a=b+c;

Valid expression!
```

EX NO: 5

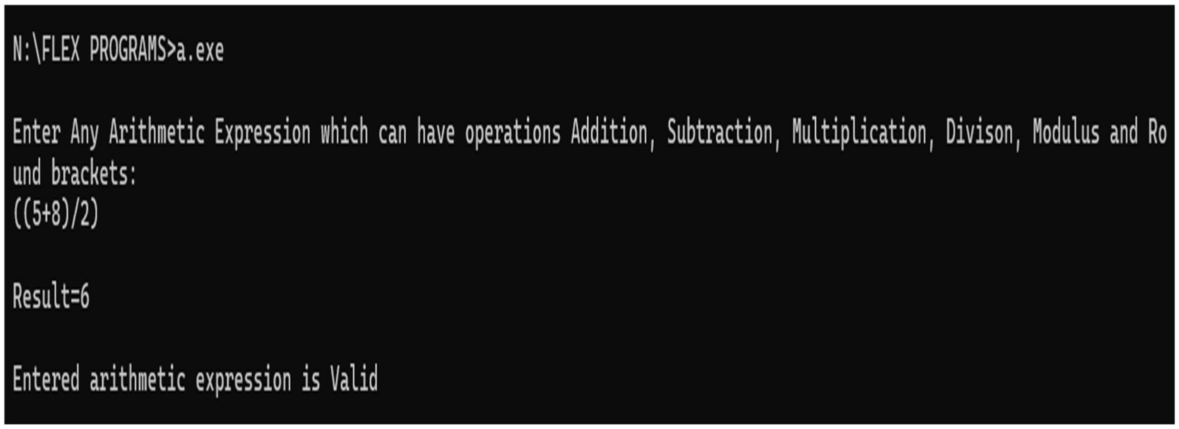
LEX PART:

```
%{
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}

%%
[0-9]+ {
        yylval=atoi(yytext);
        return NUMBER;
      }
[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
return 1;
}
```

YACC PART:

```
%{
   #include<stdio.h>
   int flag=0;


%}
%token NUMBER


%left '+' '-'
%left '*' '/' '%'
%left '(' ')'
%%
ArithmeticExpression: E{
       printf("\nResult=%d\n",$$);
       return 0;
      };
E:E'+'E {$$=$1+$3;}
 |E'-'E {$$=$1-$3;}
 |E''E {$$=$1$3;}
 |E'/'E {$$=$1/$3;}
 |E'%'E {$$=$1%$3;}

 |'('E')' {$$=$2;}

 | NUMBER {$$=$1;}

;

%%


void main()

{

   printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:\n");

   yyparse();

  if(flag==0)

   printf("\nEntered arithmetic expression is Valid\n\n");



}

void yyerror()

{

   printf("\nEntered arithmetic expression is Invalid\n\n");

   flag=1;

}
```

OUTPUT:



```
N:\FLEX PROGRAMS>a.exe

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Ro
und brackets:
((5+8)/2)

Result=6

Entered arithmetic expression is Valid
```

EX NO:6

LEX PART:

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

%%



[0-9]+? {yylval.sym=(char)yytext[0]; return NUMBER;}
[a-zA-Z]+? {yylval.sym=(char)yytext[0];return LETTER;}

\n {return 0;}
. {return yytext[0];}

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

YACC PART:

```
%{
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void ThreeAddressCode();
void triple();
void qudraple();
char AddToTable(char ,char, char);

int ind=0;//count number of lines
char temp = '1';//for t1,t2,t3.....
struct incod
{
char opd1;
char opd2;
char opr;
};
%}

%union
{
 char sym;
}

%token <sym> LETTER NUMBER
%type <sym> expr
%left '+'
```

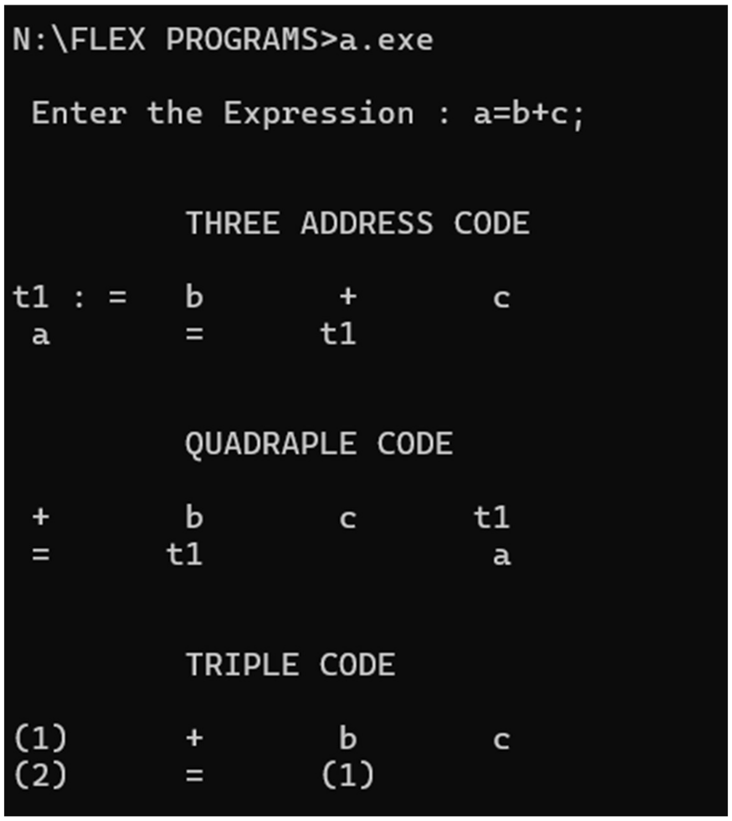
```
%left '*''/'
%left '-'
%%

statement: LETTER '=' expr ';' {AddToTable((char)$1,(char)$3,'=');}
| expr ';'
;

expr:
 expr '+' expr {$$ = AddToTable((char)$1,(char)$3,'+');}
| expr '-' expr {$$ = AddToTable((char)$1,(char)$3,'-');}
| expr '*' expr {$$ = AddToTable((char)$1,(char)$3,'*');}
| expr '/' expr {$$ = AddToTable((char)$1,(char)$3,'/');}
| '(' expr ')' {$$ = (char)$2;}
| NUMBER {$$ = (char)$1;}
| LETTER {$$ = (char)$1;}
|'-' expr {$$ = AddToTable((char)$2,(char)'\t','-');}
;

%%

yyerror(char *s)
{
 printf("%s",s);
 exit(0);
}

struct incod code[20];
```

```
char AddToTable(char opd1,char opd2,char opr)
{
 code[ind].opd1=opd1;
 code[ind].opd2=opd2;
 code[ind].opr=opr;
 ind++;
 return temp++;
}

void ThreeAddressCode()
{
 int cnt = 0;
 char temp = '1';
 printf("\n\n\t THREE ADDRESS CODE\n\n");
 while(cnt<ind)
 {
  if(code[cnt].opr != '=')
    printf("t%c : = \t",temp++);

  if(isalpha(code[cnt].opd1))
    printf(" %c\t",code[cnt].opd1);
  else if(code[cnt].opd1 >='1' && code[cnt].opd1 <='9')
    printf("t%c\t",code[cnt].opd1);

  printf(" %c\t",code[cnt].opr);

  if(isalpha(code[cnt].opd2))
    printf(" %c\n",code[cnt].opd2);
  else if(code[cnt].opd2 >='1' && code[cnt].opd2 <='9')
```

```
    printf("t%c\n",code[cnt].opd2);

  cnt++;
 }
}

void quadraple()
{
 int cnt = 0;
 char temp = '1';
 printf("\n\n\t QUADRAPLE CODE\n\n");
 while(cnt<ind)
 {
  printf(" %c\t",code[cnt].opr);
  if(code[cnt].opr == '=')
  {
   if(isalpha(code[cnt].opd2))
    printf(" %c\t \t",code[cnt].opd2);
   else if(code[cnt].opd2 >='1' && code[cnt].opd2 <='9')
    printf("t%c\t \t",code[cnt].opd2);

    printf(" %c\n",code[cnt].opd1);
   cnt++;
   continue;
  }
  if(isalpha(code[cnt].opd1))
    printf(" %c\t",code[cnt].opd1);
  else if(code[cnt].opd1 >='1' && code[cnt].opd1 <='9')
    printf("t%c\t",code[cnt].opd1);
  if(isalpha(code[cnt].opd2))
    printf(" %c\t",code[cnt].opd2);
  else if(code[cnt].opd2 >='1' && code[cnt].opd2 <='9')
    printf("t%c\t",code[cnt].opd2);
  else  printf("  %c",code[cnt].opd2);

  printf("t%c\n",temp++);

  cnt++;
 }
}

void triple()
{
 int cnt=0;
 char temp='1';
 printf("\n\n\t TRIPLE CODE\n\n");
  while(cnt<ind)
 {
  printf("(%c) \t",temp);
  printf(" %c\t",code[cnt].opr);
  if(code[cnt].opr == '=')
  {
   if(isalpha(code[cnt].opd2))
    printf(" %c \t \t",code[cnt].opd2);
   else if(code[cnt].opd2 >='1' && code[cnt].opd2 <='9')
    printf("(%c)\n",code[cnt].opd2);
   cnt++;
```

```
     temp++;
     continue;
    }
    if(isalpha(code[cnt].opd1))
      printf(" %c \t",code[cnt].opd1);
    else if(code[cnt].opd1 >='1' && code[cnt].opd1 <='9')
      printf("(%c)\t",code[cnt].opd1);

    if(isalpha(code[cnt].opd2))
      printf(" %c \n",code[cnt].opd2);
    else if(code[cnt].opd2 >='1' && code[cnt].opd2 <='9')
      printf("(%c)\n",code[cnt].opd2);
    else  printf("  %c\n",code[cnt].opd2);

    cnt++;
    temp++;
   }
  }

  main()
  {
  printf("\n Enter the Expression : ");
  yyparse();
  ThreeAddressCode();
  quadraple();
  triple();
  }
```

OUTPUT:



```
N:\FLEX PROGRAMS>a.exe

 Enter the Expression : a=b+c;


         THREE ADDRESS CODE

t1 : =   b       +       c
 a       =      t1


         QUADRAPLE CODE

 +       b       c      t1
 =      t1               a


         TRIPLE CODE

(1)      +       b       c
(2)      =      (1)
```

EX NO 7:

LEX PART:

```
%{

#include"y.tab.h"

#include<stdio.h>

#include<string.h>

int LineNo=1;

%}

identifier [a-zA-Z][_a-zA-Z0-9]*

number [0-9]+|([0-9]*\.[0-9]+)

%%

main\(\) return MAIN;

if return IF;

else return ELSE;

while return WHILE;

int |

char |

float return TYPE;

{identifier} {strcpy(yylval.var,yytext);

return VAR;}
```

```
{number} {strcpy(yylval.var,yytext);
return NUM;}
\< |
\> |
\>= |
\<= |
== {strcpy(yylval.var,yytext);
return RELOP;}
[ \t] ;
\n LineNo++;
. return yytext[0];
%%
```

YACC PART:

```
%{
#include<string.h>
#include<stdio.h>
struct quad
{
char op[5];
char arg1[10];
char arg2[10];
char result[10];

}QUAD[30];

struct stack

{


int items[100];

int top;

}stk;

int Index=0,tIndex=0,StNo,Ind,tInd;

extern int LineNo;

%}

%union

{

char var[10];

}

%token <var> NUM VAR RELOP

%token MAIN IF ELSE WHILE TYPE

%type <var> EXPR ASSIGNMENT CONDITION IFST ELSEST WHILELOOP

%left '-' '+'

%left '*' '/'

%%

PROGRAM : MAIN BLOCK

;

BLOCK: '{' CODE '}'

;

CODE: BLOCK
```

```
| STATEMENT CODE

| STATEMENT

;

STATEMENT: DESCT ';'

| ASSIGNMENT ';'

| CONDST

| WHILEST

;

DESCT: TYPE VARLIST

;

VARLIST: VAR ',' VARLIST

| VAR

;

ASSIGNMENT: VAR '=' EXPR{

strcpy(QUAD[Index].op,"=");

strcpy(QUAD[Index].arg1,$3);

strcpy(QUAD[Index].arg2,"");

strcpy(QUAD[Index].result,$1);

strcpy($$,QUAD[Index++].result);

}

;

EXPR: EXPR '+' EXPR {AddQuadruple("+",$1,$3,$$);}

| EXPR '-' EXPR {AddQuadruple("-",$1,$3,$$);}

| EXPR '*' EXPR {AddQuadruple("*",$1,$3,$$);}

| EXPR '/' EXPR {AddQuadruple("/",$1,$3,$$);}
| '-' EXPR {AddQuadruple("UMIN",$2,"",$$);}

| '(' EXPR ')' {strcpy($$,$2);}

| VAR

| NUM

;

CONDST: IFST{

Ind=pop();

sprintf(QUAD[Ind].result,"%d",Index);

Ind=pop();

sprintf(QUAD[Ind].result,"%d",Index);

}

| IFST ELSEST

;

IFST: IF '(' CONDITION ')' {

strcpy(QUAD[Index].op,"==");

strcpy(QUAD[Index].arg1,$3);

strcpy(QUAD[Index].arg2,"FALSE");

strcpy(QUAD[Index].result,"-1");

push(Index);

Index++;

}

BLOCK { strcpy(QUAD[Index].op,"GOTO"); strcpy(QUAD[Index].arg1,"");

strcpy(QUAD[Index].arg2,"");

strcpy(QUAD[Index].result,"-1");

push(Index);
Index++;

};

ELSEST: ELSE{

tInd=pop();

Ind=pop();

push(tInd);

sprintf(QUAD[Ind].result,"%d",Index);

}

BLOCK{

Ind=pop();

sprintf(QUAD[Ind].result,"%d",Index);

};

CONDITION: VAR RELOP VAR {AddQuadruple($2,$1,$3,$$);

StNo=Index-1;

}

| VAR

| NUM

;

WHILEST: WHILELOOP{

Ind=pop();

sprintf(QUAD[Ind].result,"%d",StNo);

Ind=pop();

sprintf(QUAD[Ind].result,"%d",Index);

}

;
```

```
WHILELOOP: WHILE'('CONDITION ')' {
strcpy(QUAD[Index].op,"==");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"FALSE");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
BLOCK {
strcpy(QUAD[Index].op,"GOTO");
strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
;
%%
extern FILE *yyin;
int main(int argc,char *argv[])
{
FILE *fp;
int i;
if(argc>1)
{
```

```
fp=fopen(argv[1],"r");

if(!fp)

{

printf("\n File not found");

exit(0);

}

yyin=fp;

}

yyparse();

printf("\n\n\t\t ----------------------------""\n\t\t Pos Operator \tArg1 \tArg2 \tResult" "\n\t\t--------------
-----");

for(i=0;i<Index;i++)

{

printf("\n\t\t %d\t %s\t %s\t %s\t%s",i,QUAD[i].op,QUAD[i].arg1,QUAD[i].arg2,QUAD[i].result);

}

printf("\n\t\t -----------------------");

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

void push(int data)

{ stk.top++;

if(stk.top==100)

{

printf("\n Stack overflow\n");

exit(0);

}

stk.items[stk.top]=data;

}
```

```
int pop()
{
int data;
if(stk.top==-1)
{
printf("\n Stack underflow\n");
exit(0);
}
data=stk.items[stk.top--];
return data;
}
void AddQuadruple(char op[5],char arg1[10],char arg2[10],char result[10])
{
strcpy(QUAD[Index].op,op);
strcpy(QUAD[Index].arg1,arg1);
strcpy(QUAD[Index].arg2,arg2);
sprintf(QUAD[Index].result,"t%d",tIndex++);
strcpy(result,QUAD[Index++].result);
}
yyerror()
{
printf("\n Error on line no:%d",LineNo);
}
```

OUTPUT:

Pos	Operator	Arg1	Arg2	Result
0	<	a	b	t0
1	**==**	t0	**FALSE**	**5**
2	+	**a**	**b**	**t1**
3	**==**	**t1**		**5**
4	**GOTO**			
5	<	**a**	**b**	**t2**
6	**==**	**t2**	**FALSE**	**10**
7	+	**a**	**b**	**t3**
8	**=**	**t3**		**a**
9	**GOTO**			**5**
10	**<=**	**a**	**b**	**t4**
11	**==**	**t4**	**FALSE**	**15**
12	**-**	**a**	**b**	**t5**
13	**=**	**t5**		**c**
14	**GOTO**			**17**
15	+	**a**	**b**	**t6**
16	**=**	**t6**		**c**

EX NO: 8

```
#include<stdio.h>

#include<string.h>

#include<ctype.h>

void input();

void output();

void change(int p,int q,char *res);

void constant();

void expression();

struct expr

{

char op[2],op1[5],op2[5],res[5];

int flag;

}arr[10];

int n;

int main()

{

int ch=0;
```

```
input();
constant();
expression();
output();
}
void input()
{
int i;
printf("\n\nEnter the maximum number of expressions:");
scanf("%d",&n);
printf("\nEnter the input : \n");
for(i=0;i<n;i++)
{
scanf("%s",arr[i].op);
scanf("%s",arr[i].op1);
scanf("%s",arr[i].op2);
scanf("%s",arr[i].res);
arr[i].flag=0;
}
```

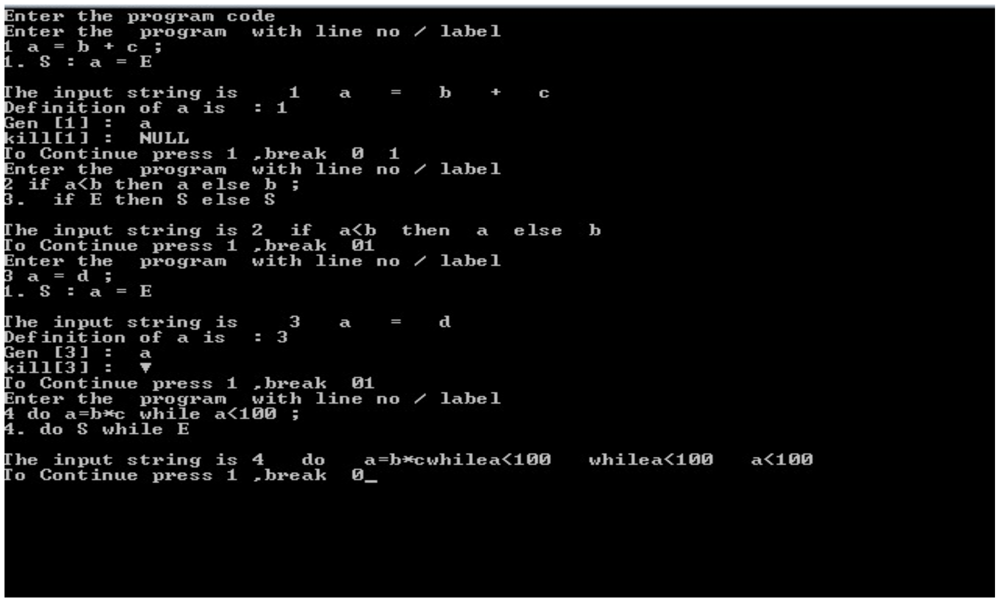
```
}
void constant()
{
int i;
int op1,op2,res;
char op,res1[5];
for(i=0;i<n;i++)
{
if(isdigit(arr[i].op1[0]) && isdigit(arr[i].op2[0]))
{
op1=atoi(arr[i].op1);
op2=atoi(arr[i].op2);
op=arr[i].op[0];
switch(op)
{
case '+':
res=op1+op2;
break;
case '-':
```

```
res=op1-op2;

break;

case '*':

res=op1*op2;

break;

case '/':

res=op1/op2;

break;

}

sprintf(res1,"%d",res);

arr[i].flag=1;

change(i,i,res1);

}

}

}

void expression()

{

int i,j;

for(i=0;i<n;i++)
```

```
{
for(j=i+1;j<n;j++)
{
if(strcmp(arr[i].op,arr[j].op)==0)
{
if(strcmp(arr[i].op,"+")==0||strcmp(arr[i].op,"*")==0)
{
if(strcmp(arr[i].op1,arr[j].op1)==0&&strcmp(arr[i].op2,arr[j].op2)==0 ||
strcmp(arr[i].op1,arr[j].op2)==0&&strcmp(arr[i].op2,arr[j].op1)==0)
{
arr[j].flag=1;
change(i,j,NULL);
}
}
else
{
if(strcmp(arr[i].op1,arr[j].op1)==0&&strcmp(arr[i].op2,arr[j].op2)==0)
{
arr[j].flag=1;
change(i,j,NULL);
}       }

}       }

}       }

void output()

{

int i=0;

printf("\nOptimized code is : ");

for(i=0;i<n;i++)

{

if(!arr[i].flag)

{

printf("\n%s %s %s %s\n",arr[i].op,arr[i].op1,arr[i].op2,arr[i].res);

}

}

}

void change(int p,int q,char *res)

{

int i;

for(i=q+1;i<n;i++)

{
```

```
if(strcmp(arr[q].res,arr[i].op1)==0)

if(res == NULL)

strcpy(arr[i].op1,arr[p].res);

else

strcpy(arr[i].op1,res);

else if(strcmp(arr[q].res,arr[i].op2)==0)

if(res == NULL)

strcpy(arr[i].op2,arr[p].res);

else

strcpy(arr[i].op2,res);

}

}
```

OUTPUT:



```
Enter the program code
Enter the  program  with line no / label
1 a = b + c ;
1. S : a = E

The input string is    1    a    =    b    +    c
Definition of a is   : 1
Gen [1] :  a
kill[1] :  NULL
To Continue press 1 ,break  0  1
Enter the  program  with line no / label
2 if a<b then a else b ;
3.  if E then S else S

The input string is 2  if  a<b  then  a  else  b
To Continue press 1 ,break  01
Enter the  program  with line no / label
3 a = d ;
1. S : a = E

The input string is    3    a    =    d
Definition of a is   : 3
Gen [3] :  a
kill[3] :  ▼
To Continue press 1 ,break  01
Enter the  program  with line no / label
4 do a=b*c while a<100 ;
4. do S while E

The input string is 4   do   a=b*cwhilea<100   whilea<100   a<100
To Continue press 1 ,break  0_
```

EX NO: 9

```
#include<stdio.h>
#include<stdlib.h>
#define TRUE 1
#define FALSE 0
typedef struct Heap
{
int data;
struct Heap *next;
}
node;
node *create();
void main()
{
int choice,val;
char ans;
node *head;
void display(node *);
node *search(node *,int);
node *insert(node *);
void dele(node **);
head=NULL;
do
{
```

```
printf("\nprogram to perform various operations on heap using dynamic memory
management");
printf("\n1.create");
printf("\n2.display");
printf("\n3.insert an element in a list");
printf("\n4.delete an element from list");
printf("\n5.quit");
printf("\nenter your chioce(1-5)");
scanf("%d",&choice);
switch(choice)
{
case 1:head=create();
break;
case 2:display(head);
break;
case 3:head=insert(head);
break;
case 4:dele(&head);
break;
case 5:exit(0);
default:
printf("invalid choice,try again");
}
}
while(choice!=5);
}
```

```
node* create()
{
node *temp,*New,*head;
int val,flag;
char ans='y';
node *get_node();
temp=NULL;
flag=TRUE;
do
{
printf("\n enter the element:");
scanf("%d",&val);
New=get_node();
if(New==NULL)
printf("\nmemory is not allocated");
New->data=val;
if(flag==TRUE)
{
head=New;
temp=head;
flag=FALSE;
}
else
{
temp->next=New;
temp=New;
}
printf("\ndo you want to enter more elements?(y/n)");
}
while(ans=='y');
printf("\nthe list is created\n");
return head;
}
node *get_node()
{
node *temp;
temp=(node*)malloc(sizeof(node));
temp->next=NULL;
return temp;
}
void display(node *head)
{
node *temp;
temp=head;
if(temp==NULL)
{
printf("\nthe list is empty\n");
return;
}
while(temp!=NULL)
{
printf("%d->",temp->data);
temp=temp->next;
}
```

```
printf("NULL");
}
node *search(node *head,int key)
{
node *temp;
int found;
temp=head;
if(temp==NULL)
{
printf("the linked list is empty\n");
return NULL;
}
found=FALSE;
while(temp!=NULL && found==FALSE)
{
if(temp->data!=key)
temp=temp->next;
else
found=TRUE;
}
if(found==TRUE)
{
printf("\nthe element is present in the list\n");
return temp;
}
else
```

```
{
printf("the element is not present in the list\n");
return NULL;
}
}
node *insert(node *head)
{
int choice;
node *insert_head(node *);
void insert_after(node *);
void insert_last(node *);
printf("n1.insert a node as a head node");
printf("n2.insert a node as a head node");
printf("n3.insert a node at intermediate position in t6he list");
printf("\nenter your choice for insertion of node:");
scanf("%d",&choice);
switch(choice)
{
case 1:head=insert_head(head);
break;
case 2:insert_last(head);
break;
case 3:insert_after(head);
break;
}
return head;
```

```
}
node *insert_head(node *head)
{
node *New,*temp;
New=get_node();
printf("\nEnter the element which you want to insert");
scanf("%d",&New->data);
if(head==NULL)
head=New;
else
{
temp=head;
New->next=temp;
head=New;
}
return head;
}
void insert_last(node *head)
{
node *New,*temp;
New=get_node();
printf("\nenter the element which you want to insert");
scanf("%d",&New->data);
if(head==NULL)
head=New;
else
```

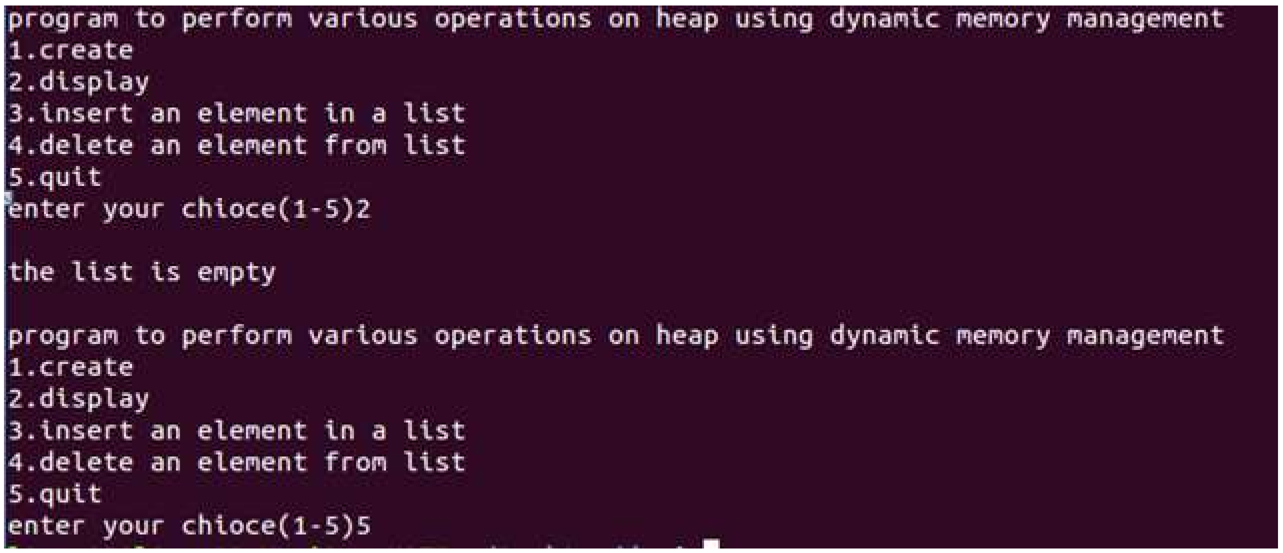
```
{
temp=head;
while(temp->next!=NULL)
temp=temp->next;
temp->next=New;
New->next=NULL;
}
}
void insert_after(node *head)
{
int key;
node *New,*temp;
New=get_node();
printf("\nenter the elements which you want to insert");
scanf("%d",&New->data);
if(head==NULL)
{
head=New;
}
else
{
printf("\enter the element which you want to insert the node");
scanf("%d",&key);
temp=head;
do
{
```

```
if(temp->data==key)
{
New->next-temp->next;
temp->next=New;
return;
}
else
temp=temp->next;
}
while(temp!=NULL);
}
}
node *get_prev(node *head,int val)
{
node *temp,*prev;
int flag;
temp=head;
if(temp==NULL)
return NULL;
flag=FALSE;
prev=NULL;
while(temp!=NULL && ! flag)
{
if(temp->data!=val)
{
prev=temp;
```

```
temp=temp->next;
}
else
flag=TRUE;
}
if(flag)
return prev;
else
return NULL;
}
void dele(node **head)
{
node *temp,*prev;
int key;
temp=*head;
if(temp==NULL)
{
printf("\nthe list is empty\n");
return;
}
printf("\nenter the element you want to delete:");
scanf("%d",&key);
temp=search(*head,key);
if(temp!=NULL)
{
prev=get_prev(*head,key);
```

```
if(prev!=NULL)
{
prev->next=temp->next;
free(temp);
}
else
{
*head=temp->next;
free(temp);
}
printf("\nthe element is deleted\n");

}
}
```

OUTPUT:

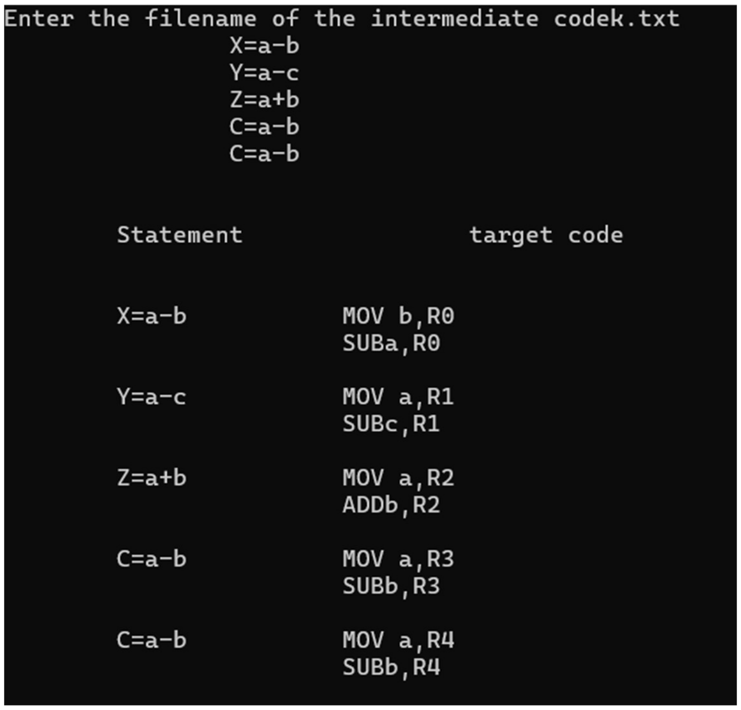


```
program to perform various operations on heap using dynamic memory management
1.create
2.display
3.insert an element in a list
4.delete an element from list
5.quit
enter your chioce(1-5)2

the list is empty

program to perform various operations on heap using dynamic memory management
1.create
2.display
3.insert an element in a list
4.delete an element from list
5.quit
enter your chioce(1-5)5
```

EX NO: 10

```
#include <stdio.h >
#include <stdio.h >
#include<conio.h>
#include <string.h >
void main() {
char icode[10][30], str[20], opr[10];
int i = 0;
clrscr();
printf("In Enter the set of intermediate code (terminated by exit):\n");
do
1
scanf("%s", icode[i]);
} while (strcmp(icode[i++], "exit") != 0);
printf("In target 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 7:
strcpy(opr, "DIV ");
break;
printf("InitMov %c,R%d", str[2], i);
printf("Init%s%c,R%d", opr, str[4], i);
printf("InitMov R%d,%c", i, str[0]);
} while (strcmp(icode[++i], "exit") != 0);
getch();
```

OUTPUT:



```
Enter the filename of the intermediate codek.txt
                X=a-b
                Y=a-c
                Z=a+b
                C=a-b
                C=a-b


        Statement                    target code


        X=a-b           MOV b,R0
                        SUBa,R0

        Y=a-c           MOV a,R1
                        SUBc,R1

        Z=a+b           MOV a,R2
                        ADDb,R2

        C=a-b           MOV a,R3
                        SUBb,R3

        C=a-b           MOV a,R4
                        SUBb,R4
```

EX NO: 11

```
#include<stdio.h> #include<conio.h> #include<string.h> struct op
{
char l; char r[20];
}op[10],pr[10];
void main()
{
int a,i,k,j,n,z=0,m,q;

char *p,*l; char temp,t; char *tem; clrscr();
printf("enter no of values"); scanf("%d",&n); for(i=0;i<n;i++)
{
printf("left\t"); op[i].l=getche();
printf("right:\t"); scanf("%s",op[i].r);

}
printf("intermediate Code\n") ; for(i=0;i<n;i++)
{
printf("%c=",op[i].l);
printf("%s\n",op[i].r);
}
for(i=0;i<n-1;i++)
{
temp=op[i].l; for(j=0;j<n;j++)
{
```

```
p=strchr(op[j].r,temp); if(p)
{
pr[z].l=op[i].l; strcpy(pr[z].r,op[i].r); z++ ;
}} }
pr[z].l=op[n-1].l;
strcpy(pr[z].r,op[n-1].r); z++;
printf("\nafter dead code elimination\n"); for(k=0;k<z;k++)
{
printf("%c\t=",pr[k].l);
printf("%s\n",pr[k].r);
}
//sub expression elimination for(m=0;m<z;m++)
{
tem=pr[m].r; for(j=m+1;j<z;j++)
{
p=strstr(tem,pr[j].r); if(p)
{
t=pr[j].l; pr[j].l=pr[m].l  ; for(i=0;i<z;i++)
{
l=strchr(pr[i].r,t) ; if(l)
{
a=l-pr[i].r;
//printf("pos: %d",a);
pr[i].r[a]=pr[m].l;
}}}}}
printf("eliminate common expression\n");
```

```
for(i=0;i<z;i++)
{
printf("%c\t=",pr[i].l);
printf("%s\n",pr[i].r);
}
// duplicate production elimination for(i=0;i<z;i++)
{
for(j=i+1;j<z;j++)
{
q=strcmp(pr[i].r,pr[j].r);
if((pr[i].l==pr[j].l)&&!q)
{
pr[i].l='\0'; strcpy(pr[i].r,'\0');
}}
}
printf("optimized code"); for(i=0;i<z;i++)
{
if(pr[i].l!='\0')
{
printf("%c=",pr[i].l);
printf("%s\n",pr[i].r);
}
}
getch();
}
```

OUTPUT:

```
enter no of values 5
left	aright: 9
left	bright: c+d
left	eright: c+d
left	fright: b+e
left	rright: f
intermediate Code
a=9
b=c+d
e=c+d
f=b+e
r=f

after dead code elimination
b	=c+d
e	=c+d
f	=b+e
r	=f


eliminate common expression
b	=c+d
b	=c+d
f	=b+b
r	=f
optimized code
b=c+d
f=b+b
r=f
```