```
/*IMPLEMENTATION OF SYMBOL TABLE*/
#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("Symbol \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++;
}}}}
```

OUTPUT :

```
10.0.0.11 - PuTTY
login as: exam01
exam01@10.0.0.11's password:
Last login: Sat Aug 20 12:31:06 2022 from 10.0.24.18
-bash-4.2$ vi symtab.c
-bash-4.2$ cc symtab.c
-bash-4.2$ ./a.out
Expression terminated by $:c=a+b$
Given Expression:c=a+b
 Symbol Table
Symbol    addr     type
c         167190536        identifier

 =        167190640        operator

a         167190712        identifier

 +        167190816        operator

b         167190864        identifier
-bash-4.2$
```

```
/*Lexical analyser to recognize a few pattern in C*/

#include<stdio.h>
#include<ctype.h>
#include<string.h>
void main()
{
FILE *fi,*fo,*fop,*fk;
int flag=0,i=1;
char c,t,a[15],ch[15],file[20];
printf("\n Enter the File Name:");
scanf("%s",&file);
fi=fopen(file,"r");
fo=fopen("inter.c","w");
fop=fopen("oper.c","r");
fk=fopen("key.c","r");
c=getc(fi);
while(!feof(fi))
{
if(isalpha(c)||isdigit(c)||(c=='['||c==']'||c=='.'==1))
fputc(c,fo);
else
{
if(c=='\n')
fprintf(fo,"\t$\t");
else fprintf(fo,"\t%c\t",c);
}
c=getc(fi);
}
fclose(fi);
fclose(fo);
fi=fopen("inter.c","r");
printf("\n Lexical Analysis");
fscanf(fi,"%s",a);
printf("\n Line: %d\n",i++);
while(!feof(fi))
{
if(strcmp(a,"$")==0)
{
printf("\n Line: %d \n",i++);
fscanf(fi,"%s",a);
}
fscanf(fop,"%s",ch);
while(!feof(fop))
{
if(strcmp(ch,a)==0)
{
```

```
fscanf(fop,"%s",ch);
printf("\t\t%s\t:\t%s\n",a,ch);
flag=1; } fscanf(fop,"%s",ch);
}
rewind(fop);
fscanf(fk,"%s",ch);
while(!feof(fk))
{
if(strcmp(ch,a)==0)
{
fscanf(fk,"%k",ch);
printf("\t\t%s\t:\tKeyword\n",a);
flag=1;
}
fscanf(fk,"%s",ch);
}
rewind(fk);
if(flag==0)
{
if(isdigit(a[0]))
printf("\t\t%s\t:\tConstant\n",a);
else
printf("\t\t%s\t:\tIdentifier\n",a);
}
flag=0;
fscanf(fi,"%s",a); }
}
```

---

vi key.c

```
int
void
main
char
if
for
while
else
printf
scanf
FILE
Include
stdio.h
conio.h
iostream.h
```

---

vi oper.c
(open para
) closepara
{ openbrace
} closebrace
< lesser
> greater
" doublequote ' singlequote
: colon
; semicolon
preprocessor
= equal
== asign
% percentage
^ bitwise
& reference
* star
+ add
- sub
\ backslash
/ slash

---

vi.sample.c

```
#include<stdio.h>
Void main()
{
int a;
a=2+3;
printf("result is %d",a);\
}
```

```
[exam24@mahendralinux ~]$ vi cprg.c
[exam24@mahendralinux ~]$ vi sample.c
[exam24@mahendralinux ~]$ cc cprg.c
[exam24@mahendralinux ~]$ ./a.out

 Enter the File Name:sample.c

 Lexical Analysis
 Line: 1
                #       :       preprocessor
                #       :       preprocessor
                include :       Identifier
                <       :       lesser
                <       :       lesser
                stdio.h :       Keyword
                >       :       greater
                >       :       greater

 Line: 2
                $       :       Identifier
```

```
/*Implementation of lexical Analyser using Lex tool*/

%{
int COMMENT=0;
%}
identifier [a-zA-Z][a-z_A-Z0-9]*
%%
#.* {printf("\n%s is a preprocessor directive",yytext);}
int |
float |
char |
double |
while |
for |
do |
if |
break |
continue |
void |
switch |
case |
long |
struct |
const |
typedef |
return |
else |
goto  {printf("\n%s is a KEYWORD",yytext);}
"/*" {COMMENT=1;}
"*/" {COMMENT=0;}
{identifier}\( {if(!COMMENT)printf("\n\n FUNCTION\n\t%s",yytext);}
\{ {if(!COMMENT) printf("\n BLOCK BEGINS");}
\} {if(!COMMENT) printf("\n BLOCK ENDS");}
{identifier} {if(!COMMENT) printf("\n%s is an IDENTIFIER",yytext);}
\".*\" {if(!COMMENT)printf("\n\t%s is a STRING",yytext);}
[0-9]+ {if(!COMMENT)printf("\n\t%s is a NUMBER",yytext);}
\)(\;)? {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)
{
```

```
if(argc>1)
 {
  FILE *file;
  file=fopen(argv[1],"r");
  if(!file)
  {
  printf("could not open %s \n",argv[1]);
  exit(0);
  }
  yyin=file;
  }
 yylex();
 printf("\n\n");
 return 0;
 }
 int yywrap()
 {
return 0;
 }
```

OUTPUT

```
login as: exam01
exam01@10.0.0.11's password:
Last login: Sat Aug 20 14:57:58 2022 from 10.0.24.18
-bash-4.2$ vi lexprg.l
-bash-4.2$ lex lexprg.l
-bash-4.2$ cc lexprg.l
/usr/bin/ld:lexprg.l: file format not recognized; treating as linker script
/usr/bin/ld:lexprg.l:2: syntax error
collect2: ld returned 1 exit status
-bash-4.2$ cc lex.yy.c
-bash-4.2$ ./a.out newprg.c

#include<stdio.h> is a preprocessor directive

#include<conio.h> is a preprocessor directive

void is a KEYWORD

 FUNCTION
        main(
        )


 BLOCK BEGINS

int is a KEYWORD
a is an IDENTIFIER,
b is an IDENTIFIER,
c is an IDENTIFIER:

a is an IDENTIFIER
        = is an ASSIGNMENT OPERATOR
        10 is a NUMBER,
b is an IDENTIFIER
        = is an ASSIGNMENT OPERATOR
        20 is a NUMBER;

c is an IDENTIFIER
        = is an ASSIGNMENT OPERATOR
a is an IDENTIFIER+
b is an IDENTIFIER;


 FUNCTION
        printf(
        "%d" is a STRING,
c is an IDENTIFIER
        );


 BLOCK ENDS
```

/* YACC program to recognize valid arithmetic expression*/

LEX PART:

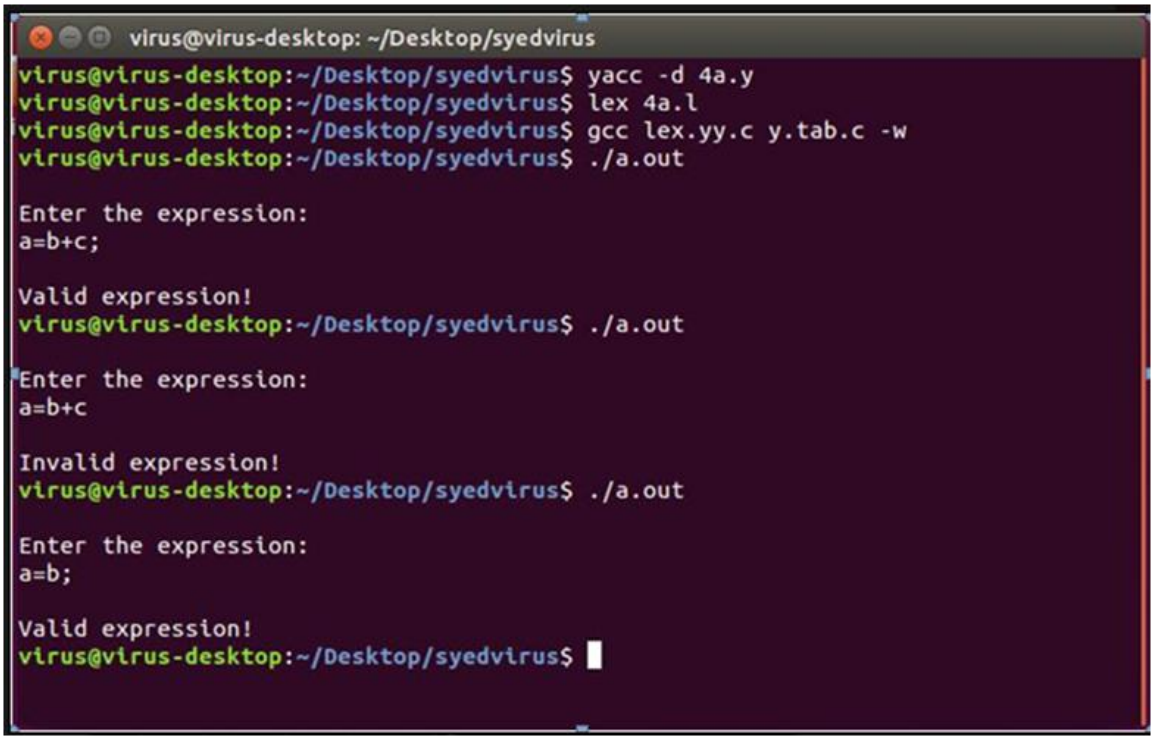
```
%{
  #include "y.tab.h"
%}
%%
[a-zA-Z_][a-zA-Z_0-9]* return id;
[0-9]+(\.[0-9]*)?
  return num;
[+/*]
 return op;
 return yytext[0];
\n
 return 0;
%%
int yywrap()
{
return 1;
}
```

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



//write code to generate Abstract Syntax Tree

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);
}
```

INPUT:

```
main()
{
int a,b,c;
if(a<b)
{
a=a+b;
}
while(a<b)
{
a=a+b;
}
if(a<=b)
{
c=a-b;
}
else
{
c=a+b;
}
}
```

OUTPUT:

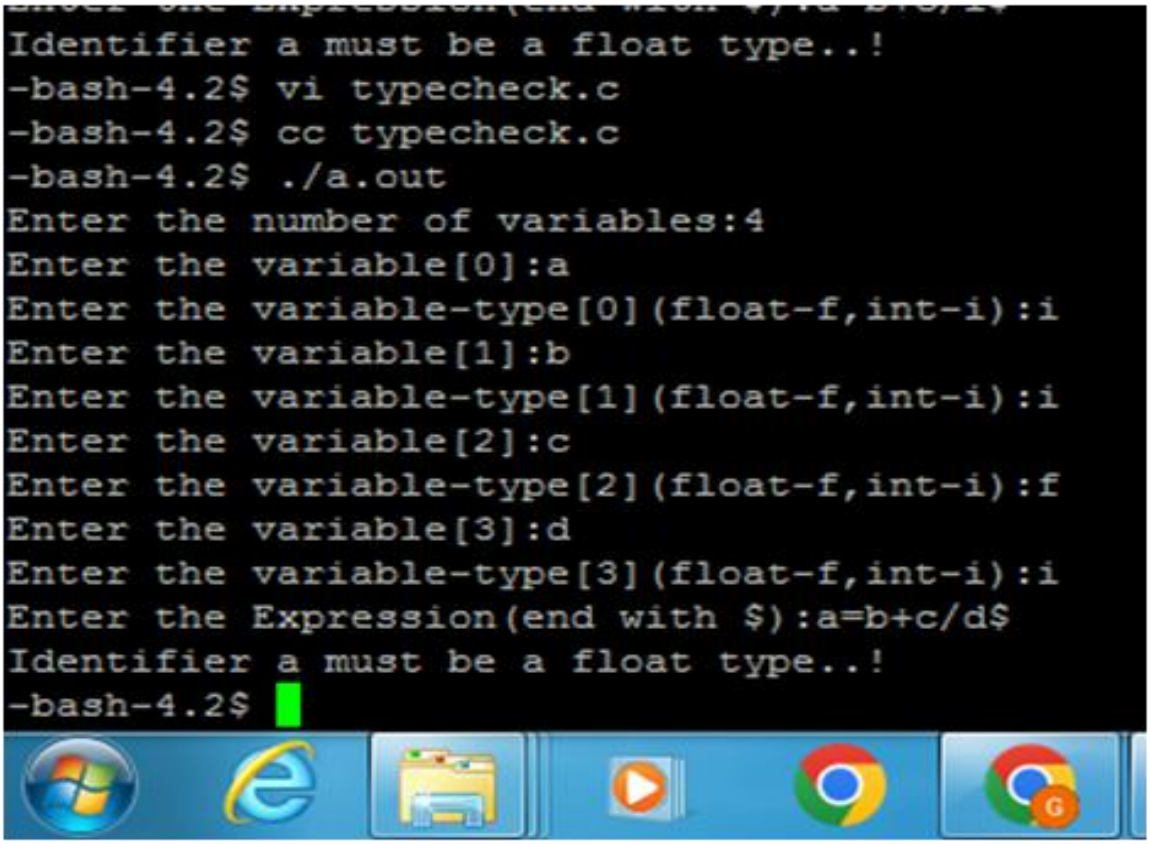
```
virus@virus-desktop: ~/Desktop/syedvirus
virus@virus-desktop:~/Desktop/syedvirus$ lex 5.l
virus@virus-desktop:~/Desktop/syedvirus$ yacc -d 5.y
virus@virus-desktop:~/Desktop/syedvirus$ gcc lex.yy.c y.tab.c -ll -lm -w
virus@virus-desktop:~/Desktop/syedvirus$ ./a.out test.c


                ----------------------------
                Pos Operator   Arg1    Arg2    Result
                --------------------
                0       <       a       b       t0
                1       ==      t0      FALSE   5
                2       +       a       b       t1
                3       =       t1              a
                4       GOTO                    5
                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
                ----------------------

virus@virus-desktop:~/Desktop/syedvirus$
```

/* Implementation of Type Checking*/

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int n,i,k,flag=0;
char vari[15],typ[15],b[15],c;
printf("Enter the number of variables:");
scanf(" %d",&n);
for(i=0;i<n;i++)
{
printf("Enter the variable[%d]:",i);
scanf(" %c",&vari[i]);
printf("Enter the variable-type[%d](float-f,int-i):",i);
scanf(" %c",&typ[i]);
if(typ[i]=='f')
flag=1;
}
printf("Enter the Expression(end with $):");
i=0;
getchar();
while((c=getchar())!='$')
{
b[i]=c;
i++;  }
k=i;
for(i=0;i<k;i++)
{
if(b[i]=='/')
{
flag=1;
break;  }  }
for(i=0;i<n;i++)
{
if(b[0]==vari[i])
{
if(flag==1)
{
if(typ[i]=='f')
{  printf("\nthe datatype is correctly defined..!\n");
break;  }
else
{  printf("Identifier %c must be a float type..!\n",vari[i]);
break;  }  }
else
{  printf("\nthe datatype is correctly defined..!\n");
```

```
break;  }  }
}
return 0;
}
```

OUTPUT



```
/* Control Flow Analysis */

#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:

```
virus@virus-desktop: ~/Desktop/syedvirus
virus@virus-desktop:~/Desktop/syedvirus$ gcc exp7b.c -w
virus@virus-desktop:~/Desktop/syedvirus$ ./a.out


Enter the maximum number of expressions:5

Enter the input :
+ 4 2 t1
+ a t1 t2
- b a t3
+ a 6 t4
* t3 t4 t5

Optimized code is :
+ a 6 t2

- b a t3

* t3 t2 t5
virus@virus-desktop:~/Desktop/syedvirus$
```

```
/* Any Storage Allocation –Stack*/
#include <stdio.h>
#include <stdlib.h>
 struct node
{
    int info;
    struct node *ptr;
}*top,*top1,*temp;
 int topelement();
void push(int data);
void pop();
void empty();
void display();
void destroy();
void stack_count();
void create();
 int count = 0;
 void main()
{
    int no, ch, e;
    printf("\n 1 - Push");
    printf("\n 2 - Pop");
    printf("\n 3 - Top");
    printf("\n 4 - Empty");
    printf("\n 5 - Exit");
    printf("\n 6 - Dipslay");
    printf("\n 7 - Stack Count");
    printf("\n 8 - Destroy stack");
     create();
     while (1)
    {
        printf("\n Enter choice : ");
        scanf("%d", &ch);
         switch (ch)
        {
        case 1:
            printf("Enter data : ");
            scanf("%d", &no);
            push(no);
            break;
        case 2:
            pop();
            break;
        case 3:
            if (top == NULL)
                printf("No elements in stack");
```

```
            else
            {
                e = topelement();
                printf("\n Top element : %d", e);
            }
            break;
        case 4:
            empty();
            break;
        case 5:
            exit(0);
        case 6:
            display();
            break;
        case 7:
            stack_count();
            break;
        case 8:
            destroy();
            break;
        default :
            printf(" Wrong choice, Please enter correct choice  ");
            break;
        }
    }
}
 /* Create empty stack */
void create()
{
    top = NULL;
}
 /* Count stack elements */
void stack_count()
{
    printf("\n No. of elements in stack : %d", count);
}
 /* Push data into stack */
void push(int data)
{
    if (top == NULL)
    {
        top =(struct node *)malloc(1*sizeof(struct node));
        top->ptr = NULL;
        top->info = data;
    }
    else
```

```
    {
        temp =(struct node *)malloc(1*sizeof(struct node));
        temp->ptr = top;
        temp->info = data;
        top = temp;
    }
         count++;
}
 /* Display stack elements */
void display()
{
    top1 = top;

    if (top1 == NULL)
    {
        printf("Stack is empty");
        return;
    }
     while (top1 != NULL)
    {
        printf("%d ", top1->info);
        top1 = top1->ptr;
    }
 }
 /* Pop Operation on stack */
void pop()
{
    top1 = top;
     if (top1 == NULL)
    {
        printf("\n Error : Trying to pop from empty stack");
        return;
    }
    else
    top1 = top1->ptr;
    printf("\n Popped value : %d", top->info);
    free(top);
    top = top1;
    count--;
}
 /* Return top element */
int topelement()
{
    return(top->info);
}
 /* Check if stack is empty or not */
```

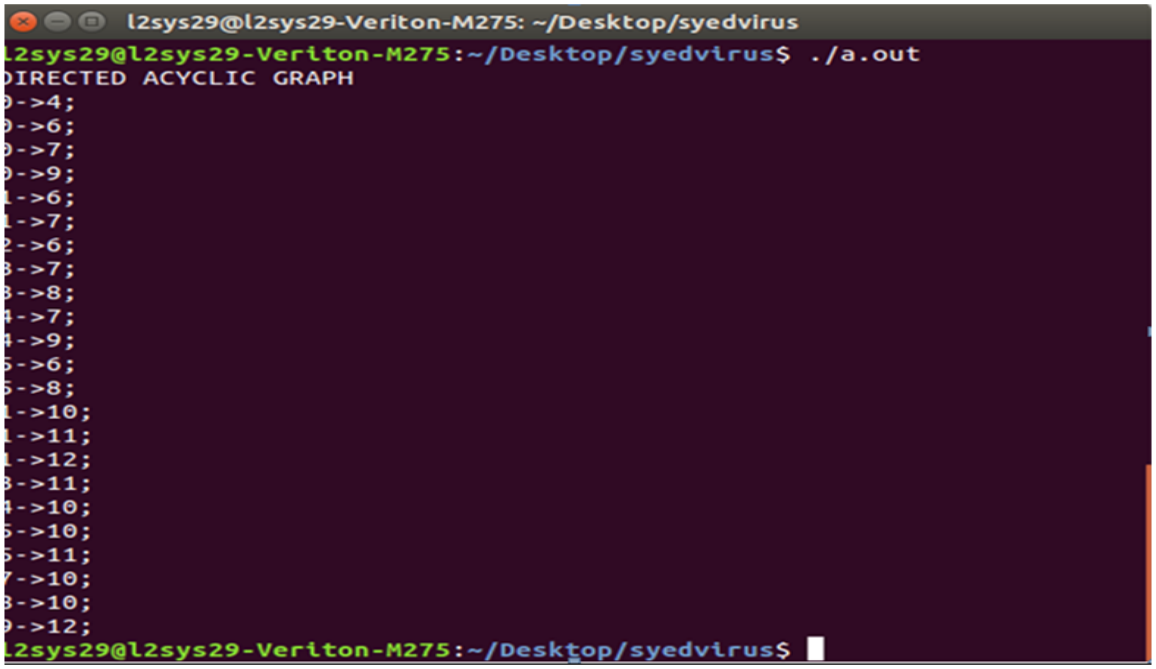
```
void empty()
{
   if (top == NULL)
      printf("\n Stack is empty");
   else
      printf("\n Stack is not empty with %d elements", count);
}
 /* Destroy entire stack */
void destroy()
{
   top1 = top;
    while (top1 != NULL)
   {
      top1 = top->ptr;
      free(top);
      top = top1;
      top1 = top1->ptr;
   }
   free(top1);
   top = NULL;
    printf("\n All stack elements destroyed");
   count = 0;
}
```

OUTPUT:

```
10.0.0.11 - PuTTY
login as: exam01
exam01@10.0.0.11's password:
Last login: Tue Nov  8 09:39:30 2022 from 10.0.24.77
-bash-4.2$ vi stack.c
-bash-4.2$ cc stack.c
-bash-4.2$ ./a.out

 1 - Push
 2 - Pop
 3 - Top
 4 - Empty
 5 - Exit
 6 - Dipslay
 7 - Stack Count
 8 - Destroy stack
 Enter choice : 1
Enter data : 10

 Enter choice : 1
Enter data : 20

 Enter choice : 1
Enter data : 30

 Enter choice : 2

 Popped value : 30
 Enter choice : 6
20 10
 Enter choice : 3

 Top element : 20
 Enter choice : 1
Enter data : 50

 Enter choice : 6
50 20 10
 Enter choice : 7

 No. of elements in stack : 3
 Enter choice :
```

```
/*Construction of DAG*/
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define MIN_PER_RANK 1
#define MAX_PER_RANK 5
#define MIN_RANKS 3
#define MAX_RANKS 5
#define PERCENT 30
void main()
{
int i,j,k,nodes=0;
srand(time(NULL));
int ranks=MIN_RANKS+(rand()%(MAX_RANKS-MIN_RANKS+1));
printf("DIRECTED ACYCLIC GRAPH\n");
for(i=1;i<ranks;i++)
{
int new_nodes=MIN_PER_RANK+(rand()%(MAX_PER_RANK-MIN_PER_RANK+1));
for(j=0;j<nodes;j++)
for(k=0;k<new_nodes;k++)
if((rand()%100)<PERCENT)
printf("%d->%d;\n",j,k+nodes);
nodes+=new_nodes;
}
}
```

OUTPUT:



/*Implementation of Simple Code Optimization Techniques*/

PROGRAM

```
#include<stdio.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;
printf("Enter the Number of Values:");
scanf("%d",&n);
for(i=0;i<n;i++)
{ printf("left: ");
scanf(" %c",&op[i].l);
printf("right: ");
 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); }
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\n",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);
} 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'; } } }
printf("Optimized Code\n");
for(i=0;i<z;i++)
{ if(pr[i].l!='\0')
{ printf("%c=",pr[i].l);
printf("%s\n",pr[i].r);
} } }
```

Output:

```
Enter the Number of Values:3
left: a
right: b+c
left: d
right: b+c
left: l
right: d
Intermediate Code
a=b+c
d=b+c
l=d

After Dead Code Elimination
d       =b+c
l       =d
Eliminate Common Expression
d       =b+c
l       =d
Optimized Code
d=b+c
l=d
```