Ex.No:1 **IMPLEMENTATION OF SYMBOL TABLE**

Date:

AIM:

To write a program in C to demonstrate symbol table management using C program.

PROGRAM:

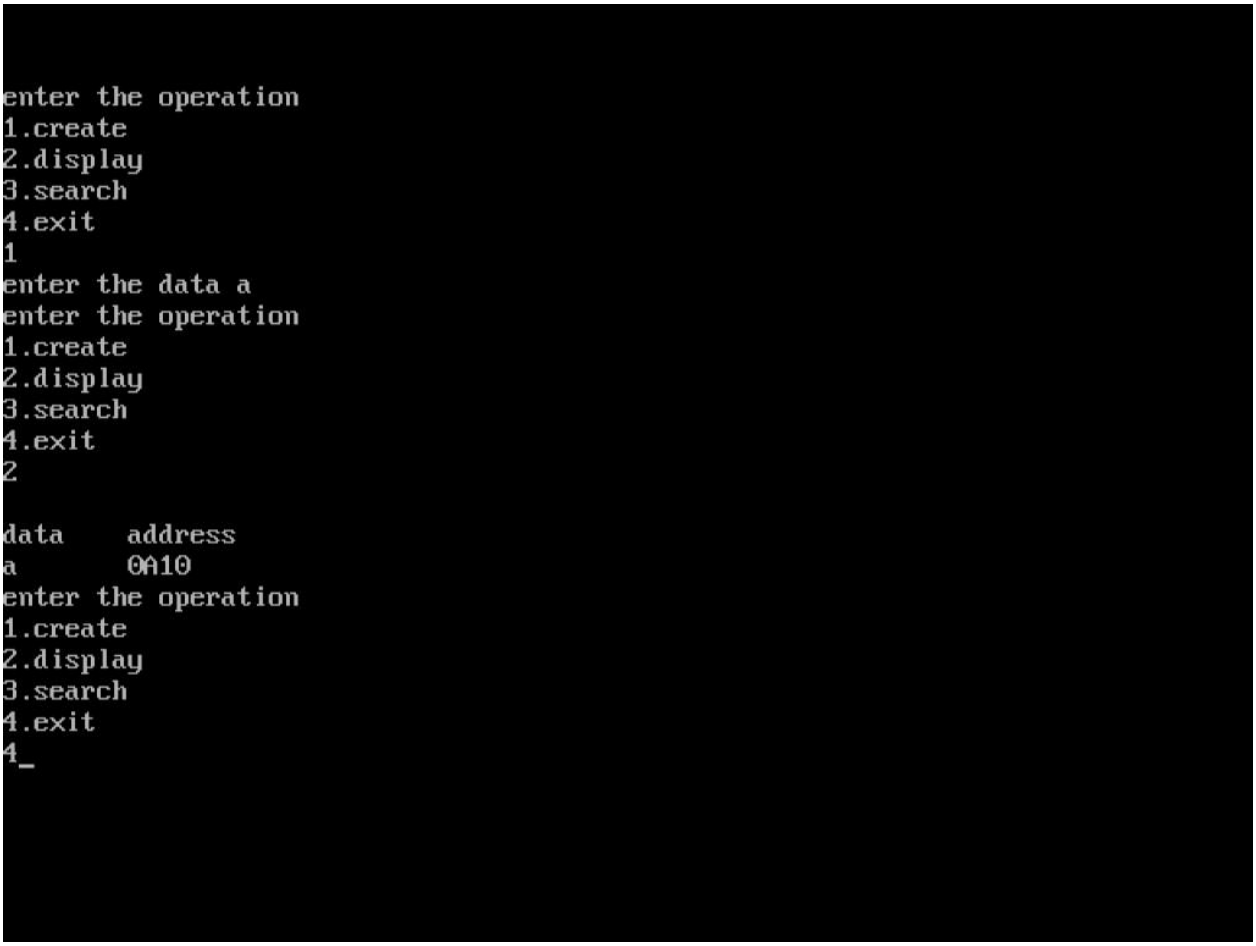
```
#include<stdio.h>
#include<malloc.h>
#include<string.h>
#include<stdlib.h>
void create();
void search();
void display();
struct node
{
        char data[500];
        struct node *nxt;
}*start,q;
void main()
{
        intch;
        clrscr();
        while(1)
        {
                printf("enter the operation\n1.create\n2.display\n3.search\n4.exit\n");
                scanf("%d",&ch);
                switch(ch)
                {
                case 1:
                        create();
```

```
                    break;
                case 2:
                    display();
                    break;
                case 3:
                    search();
                    break;
                case 4:
                    goto halt;
                default:
                    printf("enter valid choice");
                    break;
                }
        }
        halt:printf("Terminated");
}
void create()
{
        struct node *temp,*q;
        temp=malloc(sizeof(struct node));
        printf("enter the data");
        scanf("%s",temp->data);
        temp->nxt=NULL;
        if(start==NULL)
                start=temp;
        else
        {
                q=start;
                while(q->nxt!=NULL)
                        q=q->nxt;
                q->nxt=temp;
```

```
        }
}
void display()
{
        struct node *q;
        q=start;
        if(start==NULL)
        {
                printf("symbol table is empty\n");
        }
        else
        {
                printf("\ndata\taddress\n");
                while(q!=NULL)
                {
                        printf("%s\t%p\n",q->data,q);
                        q=q->nxt;
                }
        }
}
void search()
{
        struct node *q;
        char x[50];
        int count=1,flag=0;
        printf("enter the data to be searched");
        scanf("%s",x);
        q=start;
        while(q!=NULL)
        {
                if(strcmp(q->data,x)==0)
```

```
            {
                printf("element fount in %p address\n",q);
                flag=1;
                break;
            }
            q=q->nxt;
            count=count+1;
        }
        if(flag==0)
            printf("the element is not found in the list\n");
}
```

OUTPUT:



RESULT:

Thus Symbol table is implemented using data structure in C and output is verified.

Ex.No.2 **DEVELOP A LEXICAL ANALYZER TO RECOGNIZE A FEW**

Date : **PATTERNS IN C**

AIM:

To develop a lexical analyzer to identifyidentifiers, constants, comments, operators etc using C program

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
 void removeduplicate();
void final();
int Isiden(char ch);
int Isop(char ch);
int Isdel(char ch);
int Iskey(char * str);
void removeduplicate();
char op[8]={'+','-','*','/','=','<','>','%'};
char del[8]={'}','{',';','(',')','[',']',','};

char *key[]={"int","void","main","char","float"};
//char *operato[]={"+","-","/","*","<",">","=","%","<=",">=","++"};

int idi=0,idj=0,k,opi=0,opj=0,deli=0,uqdi=0,uqidi=0,uqoperi=0,kdi=0,liti=0,ci=0;
intuqdeli[20],uqopi[20],uqideni[20],l=0,j;
char uqdel[20],uqiden[20][20],uqop[20][20],keyword[20][20];
char iden[20][20],oper[20][20],delem[20],litral[20][20],lit[20],constant[20][20];
void lexanalysis(char *str)
 {
int i=0;
while(str[i]!='\0')
   {
```

```
if(Isiden(str[i]))     //for identifiers
      {
while(Isiden(str[i]))
       {
iden[idi][idj++]=str[i++];
       }
iden[idi][idj]='\0';
idi++;idj=0;
      }
else
if(str[i]=="")         //for literals
        {
lit[l++]=str[i];
for(j=i+1;str[j]!="";j++)
          {
lit[l++]=str[j];
          }
lit[l++]=str[j];lit[l]='\0';
strcpy(litral[liti++],lit);
        i=j+1;
        }
else
if(Isop(str[i]))       // for operators
         {
while(Isop(str[i]))
           {
oper[opi][opj++]=str[i++];
           }
oper[opi][opj]='\0';
opi++;opj=0;
         }
```

```
else
if(Isdel(str[i]))     //for delemeters
        {
while(Isdel(str[i]))
         {
delem[deli++]=str[i++];
         }
        }
else
        {
i++;
      }
    }

removeduplicate();
final();
}
intIsiden(char ch)
 {
if(isalpha(ch)||ch=='_'||isdigit(ch)||ch=='.')
return 1;
else return 0;
 }

intIsop(char ch)
 {
int f=0,i;
for(i=0;i<8&&!f;i++)
  {
if(ch==op[i])
    f=1;




  }
```

```
return f;
}
int Isdel(char ch)
 {
int f=0,i;
for(i=0;i<8&&!f;i++)
   {
if(ch==del[i])
     f=1;
   }
return f;
 }
intIskey(char * str)
{
inti,f=0;
for(i=0;i<5;i++)
   {
if(!strcmp(key[i],str))
      f=1;
    }
return f;
}
voidremoveduplicate()
 {
inti,j;
for(i=0;i<20;i++)
    {
uqdeli[i]=0;
uqopi[i]=0;
uqideni[i]=0;
    }
for(i=1;i<deli+1;i++)  //removing duplicate delemeters
     {
if(uqdeli[i-1]==0)
```

```
    {
uqdel[uqdi++]=delem[i-1];
for(j=i;j<deli;j++)
        {
if(delem[i-1]==delem[j])
uqdeli[j]=1;
        }
     }
    }
for(i=1;i<idi+1;i++)  //removing duplicate  identifiers
      {
if(uqideni[i-1]==0)
        {
strcpy(uqiden[uqidi++],iden[i-1]);
for(j=i;j<idi;j++)
        {
if(!strcmp(iden[i-1],iden[j]))
uqideni[j]=1;
        }
        }
   }
for(i=1;i<opi+1;i++)  //removing duplicate  operators
    {
if(uqopi[i-1]==0)
         {
strcpy(uqop[uqoperi++],oper[i-1]);
for(j=i;j<opi;j++)
           {




if(!strcmp(oper[i-1],oper[j]))
uqopi[j]=1;
           }
```

```
        }
    }
}
void final()
{
int i=0;
idi=0;
for(i=0;i<uqidi;i++)
  {
if(Iskey(uqiden[i]))       //identifying keywords
strcpy(keyword[kdi++],uqiden[i]);
else
if(isdigit(uqiden[i][0]))    //identifying constants
strcpy(constant[ci++],uqiden[i]);
else
strcpy(iden[idi++],uqiden[i]);
  }
// printing the outputs

printf("\n\tDelemeter are : \n");
for(i=0;i<uqdi;i++)
printf("\t%c\n",uqdel[i]);

printf("\n\tOperators are : \n");
for(i=0;i<uqoperi;i++)
{
printf("\t");
puts(uqop[i]);
}



printf("\n\tIdentifiers are : \n");
for(i=0;i<idi;i++)
{
```

```
 printf("\t");
 puts(iden[i]);
  }
 printf("\n\tKeywords are : \n");
 for(i=0;i<kdi;i++)
  {
 printf("\t");
 puts(keyword[i]);
  }
printf("\n\tConstants are :\n");
 for(i=0;i<ci;i++)
   {
 printf("\t");
 puts(constant[i]);
   }
 printf("\n\tLiterals are :\n");
 for(i=0;i<liti;i++)
  {
 printf("\t");
 puts(litral[i]);
  }
 }
 void main()
 {
 charstr[50];
 //clrscr();
 printf("\nEnter the string : ");
 scanf("%[^\n]c",str);



 lexanalysis(str);
 //getch();
 }
```

OUTPUT:

Enter the string:

#include<stdio.h>

void main()

{ printf (“Hello”);

}

Delimiter are :

(

)

{

;

}

Operators are :

<

>

Identifiers are :

include

stdio.h

printf

Keywords are :

Void

Main

Constants are :

Literals :

“Hello”

RESULT:

Thus the few patterns are identified in lexical analyzer

Ex. No: 3 **LEX – SAMPLE PROGRAMS**

Date:

AIM:

To write sample lex programs to illustrate basic operations.

PROGRAM 1:

Write A Lex Program To Show The Function Of Echo

```
%%
. ECHO;
\n ECHO;
%%
int yywrap(void)
{
return 1;
}
int main()
{
yylex();
return o;
}
```

OUTPUT:

```
a
a
b
b
```

PROGRAM 2:

Lex Program To Find Only Identifiers

```
ID [a-z0-9]
%%
{ID} {printf("It is an Identifier");}
%%
int yywrap(void)
{
return 1;
}
int main()
{
yylex();
return 0;
}
```

OUTPUT:

```
g
It is an Identifier
j
It is an Identifier
```

PROGRAM 3:

Lex Program To Find The Line Number Of The Given Input.

```
%{
int l=0;
%}
%%
^(.*)\n {printf("%d\t%s",l++,yytext);}
```

```
%%
int yywrap(void)
{
return 1;
}
int main(int argc, char *argv[])
{
yyin=fopen(argv[],"r");
}
```

OUTPUT:

```
Abc
0 Abc
Hai
1 Hai
```

PROGRAM 4:

Lex Program To Find The Number Of Vowels And Consonants

```
%{
int count=0;count2=0;
%}
%%
"a"|"e"|"i"|"o"|"u" {++Count; printf("\nVowel: %d",Count);}
[a-z] {++Count2; printf("Consonants: %d",Count2);}
%%
int yywrap(void)
{
return 1;
}
```

```
int main()
{
yylex();
}
```

OUTPUT:

```
a
vowel 1
b
consonants 1
```

RESULT :

Thus the sample lex programs were executed and output was verified

Ex.No:4 **IMPLEMENTATION OF LEXICAL ANALYZER USING LEX TOOL**

Date:

AIM:

To implement a lexical analyzer using Lex Tool.

PROGRAM:

```
%{
#include<stdio.h>
int e,k,c,d,i,s;
%}
%%
include |void|main|int|float|double|scanf|char|printf {printf("keyword");
i++;}[a-z][a-zA-Z0-9]* {printf("Identifier"); k++;}
[0-9]* {printf("digit"); e++;}
[+|-|*|/|=]* {printf("operator"); c++;}
[;|:|(|)|{|}|"|'|,|\n|\t]* {printf("delimeter"); d++;}
[#|<|>|%]* {printf("symbols"); s++;}
%%
int main(void)
{
yyin=fopen("lexy.txt","r");
yylex();
printf("\nidentifier %d\n",k);
printf("Symbols %d\n",s);
printf("digits %d\n",e);
printf(" Operator %d\n",c);
printf(" keywords %d\n",i);
printf("delimeter %d\n",d);

return 1;
}
```

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

INPUT:

Lexyi.txt

```
int a=10;
```

OUTPUT:

Identifier 1

Digit 1

Keyword 1

Operator 1

Delimiter 1

RESULT:

Thus a lexical analyzer is implemented using Lex Tool and the output is verified.

Ex.No:5 **PROGRAM TO RECOGNIZE A VALID ARITHMETIC EXPRESSION THAT USES OPERATOR +, - , * AND / USING YACC**

Date :

AIM:

To write a Yacc program to valid arithmetic expression using Yacc

PROGRAM:

calc.y

```
%{
   #include <stdio.h>
voidyyerror(char *);
intyylex(void);
intsym[26],i=0;
%}
%token INTEGER VARIABLE
%left '+' '-'
%left '*' '/'
%%
program:
program statement '\n'
      | /* NULL */
      ;
statement:
expression                  { printf("%d\n", $1); }
      | VARIABLE '=' expression       { printf ("Node =\t\t"); sym[$1] = $3;}
      ;
expression:
      INTEGER
      | VARIABLE                  { $$ = sym[$1]; }
      | expression '+' expression     { $$ = $1 + $3; }
      | expression '-' expression     {$$ = $1 - $3; }
      | expression '*' expression     { $$ = $1 * $3; }
      | expression '/' expression     { $$ = $1 / $3; }
```

```
        | '(' expression ')'                { $$ = $2; }
        ;
%%
voidyyerror(char *s) {
fprintf(stderr, "%s\n", s);
}
int main(void) {
yyparse();
}
```

calc.l

```
%{
   #include "y.tab.h"
   #include <stdlib.h>
voidyyerror(char *);
%}
%%
[a-z]       {
printf("Tdentifier");
return VARIABLE;
               }
[0-9]+      {
yylval = atoi(yytext);
return INTEGER;
           }
[-+()=/*\n]     { return *yytext; }
[ \t]   ;       /* skip whitespace */
.               yyerror("Unknown character");
%%
int yywrap(void)
{return 1;
```

}

OUTPUT:

```
bison -d -y calc.y
lexcalc.l
gcclex.yy.cy.tab.c
./a.out
a+b
valid
ab-
syntax error
```

RESULT:

Thus the program for validating arithmetic expression was done

Ex. No: 6 **IMPLEMENTATION OF CALCULATOR USING LEX AND YACC**

Date :

AIM :

To write a yacc program to implement calculator using yacc

PROGRAM:

valid.y

```
%{
   #include <stdio.h>
void  yyerror(char*);


int sym[26],i=0;
%}
%token INTEGER VARIABLE
%left '+' '-'
%left '*' '/'
%%
program:
program statement '\n'
      | /* NULL */
      ;
statement:
expression                  { printf("%d\n", $1); }
      | VARIABLE '=' expression       { printf ("Node =\t\t"); sym[$1] = $3;}
      ;
expression:
      INTEGER
      | VARIABLE                  { $$ = sym[$1]; }
      | expression '+' expression     { $$ = $1 + $3; }
```

```
        | expression '-' expression      {$$ = $1 - $3; }
        | expression '*' expression     { $$ = $1 * $3; }
        | expression '/' expression     { $$ = $1 / $3; }
        | '(' expression ')'                { $$ = $2; }
        ;
%%
voidyyerror(char *s) {
fprintf(stderr, "%s\n", s);
}
int main(void) {
yyparse();
}
```

valid.l

```
%{
   #include "y.tab.h"
   #include <stdlib.h>
void yyerror(char *);
%}

%%
[a-z]       {
printf("Tdentifier");
return VARIABLE;
              }
[0-9]+      {
yylval = atoi(yytext);
return INTEGER;
           }
[-+()=/*\n]     { return *yytext; }
[ \t]   ;       /* skip whitespace */
.               yyerror("Unknown character");
```

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

OUTPUT:

```
bison -d -y valid.y
lexvalid.l
gcclex.yy.cy.tab.c
./a.out
3+5
8
35/
syntax error
```

RESULT:

Thus the program for checking letter followed by letter or digits were done

Ex. No: 7 **PROGRAM TO RECOGNIZE A VALID VARIABLE WHICH STARTS WITH A LETTER FOLLOWED BY ANY NUMBER OF LETTERS OR DIGITS**

Date :

AIM :

To write a yacc program to check valid variable followed by letter or digits

PROGRAM:

valid.y

```
%{
   #include <stdio.h>
void  yyerror(char*);

int yylex(void);

int sym[26],i=0;
%}
%token INTEGER VARIABLE
%left '+' '-'
%left '*' '/'
%%
program:
program statement '\n'
      | /* NULL */
      ;
statement:
expression                  { printf("%d\n", $1); }
      | VARIABLE '=' expression       { printf ("Node =\t\t"); sym[$1] = $3;}
      ;
expression:
      INTEGER
      | VARIABLE                  { $$ = sym[$1]; }
      | expression '+' expression     { $$ = $1 + $3; }
      | expression '-' expression     {$$ = $1 - $3; }
```

```
        | expression '*' expression     { $$ = $1 * $3; }
        | expression '/' expression     { $$ = $1 / $3; }
        | '(' expression ')'            { $$ = $2; }
        ;
%%
void yyerror(char *s) {
fprintf(stderr, "%s\n", s);
}
int main(void) {
yyparse();
}
```

valid.l

```
%{
   #include "y.tab.h"
   #include <stdlib.h>
void yyerror(char *);
%}

%%
[a-z]       {
printf("Tdentifier");
return VARIABLE;
              }
[0-9]+      {
yylval = atoi(yytext);
return INTEGER;
           }
[-+()=/*\n]     { return *yytext; }
[ \t]   ;       /* skip whitespace */
.               yyerror("Unknown character");
%%
```

```
int yywrap(void)
{return 1;
}
```

OUTPUT:

```
bison -d -y valid.y
lexvalid.l
gcclex.yy.cy.tab.c
./a.out
A1
valid
10a
syntax error
```

RESULT:

Thus the program for checking letter followed by letter or digits were done

Ex.No: 8 **IMPLEMENTATION OF TYPE CHECKING**

Date:

AIM:

To write a C program to implement type checking

PROGRAM:

```
#include<stdio.h>
#include<string.h>
struct symTable
{
int type;
char var[10];
}sT[50];
int c = 0;
void sep(char a[])
        {
int len = strlen(a);
int i,j=0;
char temp[50],tp[50];
for(i = 0; i <len;++i)
{
if(a[i] != 32)
tp[i] = a[i];
else
break;
}
tp[i] = '\0';
temp[0]='\0';
++i;
for(;i <len;++i)
{
if(a[i] != ',' && a[i] != 32 && a[i] != ';')
```

```
temp[j++] = a[i];
else
{
if(strcmp(tp,"int") == 0)
sT[c].type = 1;
else if(strcmp(tp,"float") == 0)
sT[c].type = 2;
strcpy(sT[c++].var,temp);
temp[0] = '\0';
j=0;
}
}
}
int check(char a[])
{
int len =  strlen(a);
int i, j = 0,key = 0,k;
char temp[50];
for(i = 0;i <len;++i)
{
if(a[i] != 32 && a[i] != '+' && a[i] != '=' && a[i] != ';')
temp[j++] = a[i];
else
{
temp[j]='\0';
for(k = 0;k < c;++k)
{
if(strcmp(sT[k].var,temp) == 0)
{
if(key == 0)
key = sT[k].type;
else if(sT[k].type != key)
return 0;
}
```

```
}
j = 0;
}
}
return 1;
}
void main()
{
int n,ans,i;
char s[50];
printf("\n Enter the total lines of declaration\n");
scanf("%d",&N);
while(N--)
{
scanf(" %[^\n]",s);sep(s);
}
printf("Enter the expression:\n");scanf("
%[^\n]",s);
if(check(s))
printf("Correct\n");
else
printf("Semantic error\n");
}
```

OUTPUT:

```
 Enter the total lines of declaration
2

int a,b;
float c;
Enter the expression:
c=a+b;
Semantic error
```

RESULT:

Thus, the c program for implementation of type checking was done successfully.

Ex. No: 9 **IMPLEMENTATION OF THE FRONT END OF THE COMPILER**

DATE:

AIM:

To implement the front end of the compiler using C program

PROGRAM:

```
#include<stdio.h>
#include<string.h>
void main()
{
char pg[100][100],str1[24];
int tem=-1,ct=0,i=-1,j=0,j1,pos=-1,t=-1,flag,flag1,tt=0,fg=0;
printf("Enter the codings \n");
while(i>-2)
{i++;
lab1:
t++;
scanf("%s",&pg[i]);
if((strcmp(pg[i],"getch();"))==0)
{i=-2;
goto lab1;}}
printf("\n pos \t oper \t arg1 \t arg2 \t result \n");
while(j<t){
        lop:ct=0;
        if(pg[j][1]=='='){
                pos++;
                tem++;
                printf("%d \t %c \t%c \t %c \t t %d \n",pos,pg[j][3],pg[j][2],pg[j][4],tem);
                pos++;
                printf("%d \t:=  \t t %d \t \t %c \n",pos,tem,pg[j][0]);
        }
```

```
else if(((strcmp(pg[j],"if"))==0)||((strcmp(pg[j],"while"))==0))
{
if((strcmp(pg[j],"if"))==0)
strcpy(str1,"if");
if((strcmp(pg[j],"while"))==0)
strcpy(str1,"ehile");
j++;
j1=j;tem++;
pos++;
if(pg[j][3]=='=')
printf("%d \t%c \t %c \t %c \t t%d %d \n",pos,pg[j][2],pg[j][1],pg[j][4],tem);
else
printf("%d \t %c \t %c \t %c \t t%d \n",pos,pg[j][2],pg[j][1],pg[j][3],tem);
j1+=2;
pos++;
while((strcmp(pg[j],"}"))!=0)
{j++;
if(pg[j][1]== '=')
{tt=j;
fg=1;
}
ct++;
}
ct=ct+pos+1;
printf("%d \t==          \tt %d \t FALSE \t %d \n",pos,tem,ct);
if(fg==1)
{j=tt;
goto lop;
}
while((strcmp(pg[j],"}"))!=0)
{pos++;
```

```
tem++;
printf("%d \t %c \t %c \t %c \t t %d \n",pos,pg[j][3],pg[j][2],pg[j][4],tem);
pos++;
printf("%d \t := \t t %d \t \t %c \n",pos,tem,pg[j][0]);
j++;
}
if((strcmp(pg[j+1],"else"))==0)
{ct=0;
j++;
j1=j;
j1+=2;
pos++;
while((strcmp(pg[j],"}"))!=0)
{j1++;
ct++;
}ct=ct*2;
ct++;
ct+=(pos+1);
j+=2;
printf("%d \t GOTO \t \t \\t %d \n ",pos,ct);
while((strcmp(pg[j],"}" ))!=0)
{pos++;
tem++;
printf("%d \t %c \t%c \t %c \t t%d \n",pos,pg[j][3],pg[j][2],pg[j][4],tem);
pos++;
printf("%t:= \t t %d \t \t%c \n",pos,tem,pg[j][0]);
j++;
}
pos++;
printf("%d \t GOTO \t \t \t \%d \n",pos,ct);
}}
```

```
if((strcmp(pg[j],"}"))==0)

{pos++;
printf("%d \t GOTO \t \t \t %d \n",pos,pos+1);
}
j++;
}
}
```

OUTPUT:

```
Enter the codings
#include<stdio.h>
#include<conio.h>
void main()
{
int a=5,b=6,c;
clrscr();
c=a+b;
printf("%d",c);
getch();
}

 pos     oper    arg1    arg2    result
0         ,      5       b       t 0
1        :=       t 0            a
2         +      a       b       t 1
3        :=       t 1            c

--------------------------------
Process exited after 82.92 seconds with return value 11
Press any key to continue . . .
```

RESULT:

Thus the front end of the compiler is implemented in C and output is verified.

Ex.No : 10 **IMPLEMENTATION OF SIMPLE CODE OPTIMIZATION**

Date : **TECHNIQUES**

AIM:

To write a C program to eliminate dead code as code optimization

PROGRAM:

```
#include<stdio.h>
void main()
{
char all_var[10];

int ptr=0,dup=0,i,j,c,a=0;

FILE *file;
file = fopen( "test.txt" , "r");
if (file) {
while ((c=getc(file)) != EOF)
   {
        putchar(c);
        if(c==';'||c==' '||(c>='0'&&c<='9')||c=='='||c=='\n'||c=='+')
continue;
else
      {
         all_var[ptr]=c;
         ptr++;
      }
   }
printf("\n The unused variables are: ");
for(i=0;i<=ptr;i++)
{
for(j = i+1; j <= ptr; j++)
dup=1;
if(all_var[i] == all_var[j])
{
```

```
}
 if(dup==0)
 {printf("%c",all_var[i]);
 dup=0;
 }
 }
 fclose(file);
 }
 }
```

INPUT:

```
a;
b;
c;
a;
a=34;
c=78;
j=45;
r=30;
```

OUTPUT:

b is unused variable

RESULT:

Thus the program for code optimization was implemented

Ex. No:11 **IMPLEMENT ANY ONE STORAGE ALLOCATION STRATEGIES(HEAP,STACK,STATIC)**

DATE:

AIM:

To implement Stack storage allocation strategies using C program.

PROGRAM:

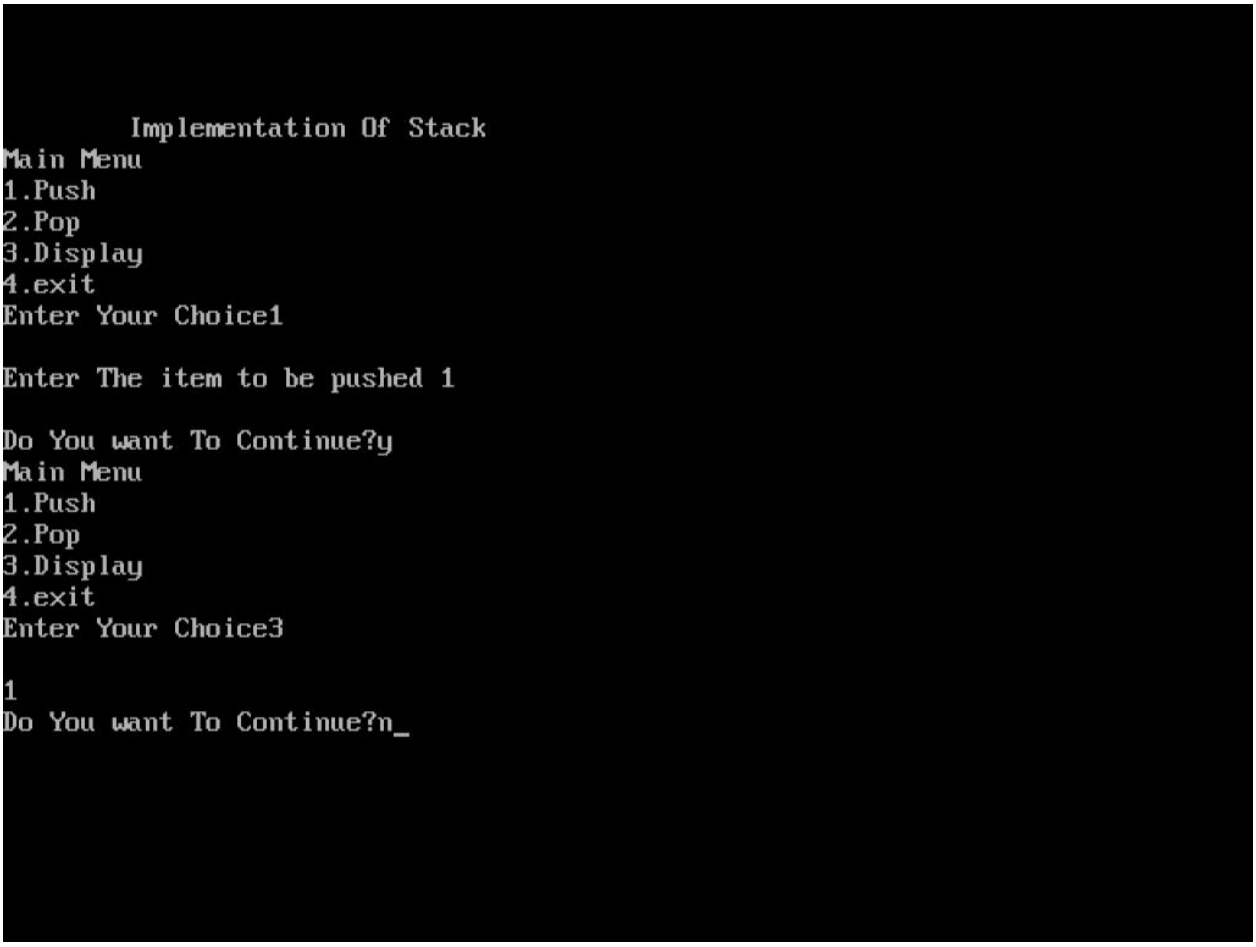
```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define size 5
struct stack
{
        int s[size];
        int top;
} st;
int stfull()
{
        if (st.top>= size - 1)
                return 1;
        else
                return 0;
}

void push(int item)
{
        st.top++;
        st.s[st.top] = item;
}
int stempty()
```

```
{
        if (st.top == -1)
                return 1;
        else
                return 0;
}
int pop()
{
        int item;
        item = st.s[st.top];
        st.top--;
        return (item);
}
void display()
{
        int i;
        if (stempty())
                printf("\nStack Is Empty!");
        else
        {
                for (i = st.top; i >= 0; i--)
                printf("\n%d", st.s[i]);
        }
}
int main()
{
        int item, choice;
        charans;
        st.top = -1;
        printf("\n\tImplementation Of Stack");
        do {
```

```
printf("\nMain Menu");
printf("\n1.Push \n2.Pop \n3.Display \n4.exit");
printf("\nEnter Your Choice");
scanf("%d", &choice);
switch (choice)
 {
case 1:
        printf("\nEnter The item to be pushed");
        scanf("%d", &item);
        if (stfull())
                printf("\nStack is Full!");
        else
                push(item);
        break;
case 2:
        if (stempty())
                printf("\nEmptystack!Underflow !!");
        else
         {
                item = pop();
                printf("\nThe popped element is %d", item);
        }
        break;
case 3:
        display();
        break;
case 4:
        goto halt;
}
printf("\nDo You want To Continue?");
ans = getche();
```

```
    } while (ans == 'Y' || ans == 'y');
    halt:
    return 0;
}
```

OUTPUT:



RESULT:

Thus the stack is implemented in C using arrays and output is verified.

Ex.No: 12 **IMPLEMENTATION OF THE BACK END OF THE COMPILER**

Date :

AIM:

To write a c program for implementing backend of a compiler.

PROGRAM:

```
#include<stdio.h>
#include<string.h>int
main(){
char inp[100][100];
int n,i,j,len;
int reg = 1;
printf("Enter the no of statements");
scanf("%d",&n);
for(i = 0; i < n; i++)
scanf("%s",&inp[i]);
for(i = 0; i < n; i++)
{
len = strlen(inp[i]);
for(j=2; j <len; j++)
{
if(inp[i][j] >= 97 &&inp[i][j] <= 122)
{
printf("LOAD R%d %c \n",reg++,inp[i][j]);
}
if(j == len-1 &&inp[i][len-j] =='=')
{
j=3;
if(inp[i][j] == '+')
{
printf("ADD R%dR%d\n",reg-2,reg-1);
```

```
printf("STORE %c R%d\n",inp[i][0],reg-2);
}
else if(inp[i][j]=='-')
{
printf("SUB R%dR%d\n",reg-2,reg-1);
printf("STORE %c R%d\n",inp[i][0],reg-2);
}
else if(inp[i][j]=='*')
{
printf("MUL R%dR%d\n",reg-2,reg-1);
printf("STORE %c R%d\n",inp[i][0],reg-2);
}
else if(inp[i][j]=='/')
{
printf("DIV R%dR%d\n",reg-2,reg-1);
printf("STORE %c R%d\n",inp[i][0],reg-2);
}
break;
}
}
}
return 0;
}
```

OUTPUT:

```
Enter the no of statements2
a=b+c;
c=c+d;
LOAD R1 b
LOAD R2 c
ADD R1 R2
STORE a R1
LOAD R3 c
LOAD R4 d
ADD R3 R4
STORE c R3
```

RESULT:

Thus, the c program for implementing backend of the compiler was done successfully.