

[illegible]

## PRACTICAL #1

**AIM : Implementation of symbol table.**

### **STEPS :**

1. Start the program.
2. Get the input from the user with the terminating symbol \$.
3. Allocate memory for the variable by dynamic memory allocation function.
4. If the next character of the symbol is an operator then only the memory is allocated.
5. While reading, the input symbol is inserted into symbol table along with its memory address.
6. The steps are repeated till \$ is reached.
7. To reach a variable, enter the variable to the searched and symbol table has been checked for corresponding variable, the variable along with its address is displayed as result.
- 8) Stop the program.

### **CODE :**

```
#include<stdio.h>
#include<iostream>
#include<ctype.h>
#include<malloc.h>
#include<string.h>
#include<math.h>

using namespace std;
int main()
{
    int i=0,j=0,x=0,n,flag=0;
    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\n\t Symbol Table\n\n");
    printf("Symbol\t Address \t Type\n");
    while(j<=n)
```

```

{
c=b[j];
if(isalpha(toascii(c)))
{
p=malloc(c);
add[x]=p;
d[x]=c;
printf("%c\t%p\t identifier\n",c,p);
x++;
}
else if(c=='+'||c=='-'||c=='*'||c=='=')
{
p=malloc(c);
add[x]=p;
d[x]=c;
printf("%c\t%p\t operator\n",c,p);
x++;
}
else if(isdigit(c))
{
p=malloc(c);
add[x]=p;
d[x]=c;
printf("%c\t%p\t constant\n",c,p);
x++;
}
j++;
}
printf("\n\nThe symbol is to be searched: ");
cin>>srch;
for(i=0;i<=x;i++)
{
if(srch==d[i])
{
printf("\nSymbol Found");
printf("\n%c%s%p\n",srch," @address ",add[i]);
flag=1;
}}
if(flag==0)
printf("\nSymbol Not Found");
getchar();

return 0;
}

```

## OUTPUT :

```
Terminal
shivam Compiler_Lab $ g++ exp1.cpp
shivam Compiler_Lab $ ./a.out
Expression terminated by $ : a + 5 * b = c $
Given Expression : a + 5 * b = c

      Symbol Table

Symbol  Address      Type
a       0x55a823603690  identifer
+       0x55a823603700  operator
5       0x55a823603740  constant
*       0x55a823603780  operator
b       0x55a8236037c0  identifer
=       0x55a823603830  operator
c       0x55a823603880  identifer

The symbol is to be searched: b

Symbol Found
b @address 0x55a8236037c0
shivam Compiler_Lab $
```

## PRACTICAL #2

**AIM : Develop a lexical analyzer to recognize a few patterns.(ex. Identifiers, constants, comments, operators etc.)**

### **STEPS :**

1. Start the program.
2. Declare all the variables and file pointers.
3. Display the input program.
4. Separate the keyword in the program and display it.
5. Display the header files of the input program
6. Separate the operators of the input program and display it.
7. Print the punctuation marks.
8. Print the constant that are present in input program.
9. Print the identifiers of the input program.

### **CODE :**

```
#include<string.h>
#include<ctype.h>
#include<stdio.h>
#include<stdlib.h>

void keyword(char str[10])
{
    if(strcmp("for",str)==0||strcmp("while",str)==0||strcmp("do",str)==0||
    strcmp("int",str)==0||strcmp("float",str)==0||strcmp("char",str)==0||
    strcmp("double",str)==0||strcmp("printf",str)==0||strcmp("switch",str)==0||
    strcmp("case",str)==0)
        printf("\n%s is a keyword",str);
    else
        printf("\n%s is an identifier",str);
}

int main()
{
    FILE *f1,*f2,*f3;
    char c,str[10],st1[10];
    int num[100],lineno=0,tokenvalue=0,i=0,j=0,k=0;
    f1=fopen("exp1.cpp","r");
    f2=fopen("identifier","w");
    f3=fopen("specialchar","w");
    while((c=getc(f1))!=EOF)
    {
        if(isdigit(c))
        {
            tokenvalue=c-'0';
            c=getc(f1);
```

```

while(isdigit(c))
{
    tokenvalue*=10+c-'0';
    c=getc(f1);
}
num[i++]=tokenvalue;
ungetc(c,f1);
}
else
if(isalpha(c))
{
    putc(c,f2);
    c=getc(f1);
    while(isdigit(c)||isalpha(c)||c=='_'||c=='$')
    {
        putc(c,f2);
        c=getc(f1);
    }
    putc(' ',f2);
    ungetc(c,f1);
}
else
if(c==' '||c=='\t')
printf(" ");
else
if(c=='\n')
lineno++;
else
putc(c,f3);
}
fclose(f2);
fclose(f3);
fclose(f1);
printf("\n the no's in the program are:");
for(j=0;j<i;j++)
    printf("\t%d",num[j]);
printf("\n");
f2=fopen("identifier","r");
k=0;
printf("the keywords and identifier are:");
while((c=getc(f2))!=EOF)
if(c!=' ')
str[k++]=c;
else
{
    str[k]='\0';
    keyword(str);
    k=0;
}
fclose(f2);
f3=fopen("specialchar","r");
printf("\n Special Characters are");

```

```

while((c=getc(f3))!=EOF)
printf("\t%c",c);
printf("\n");
fclose(f3);
printf("Total no of lines are:%d",lineno);
}

```

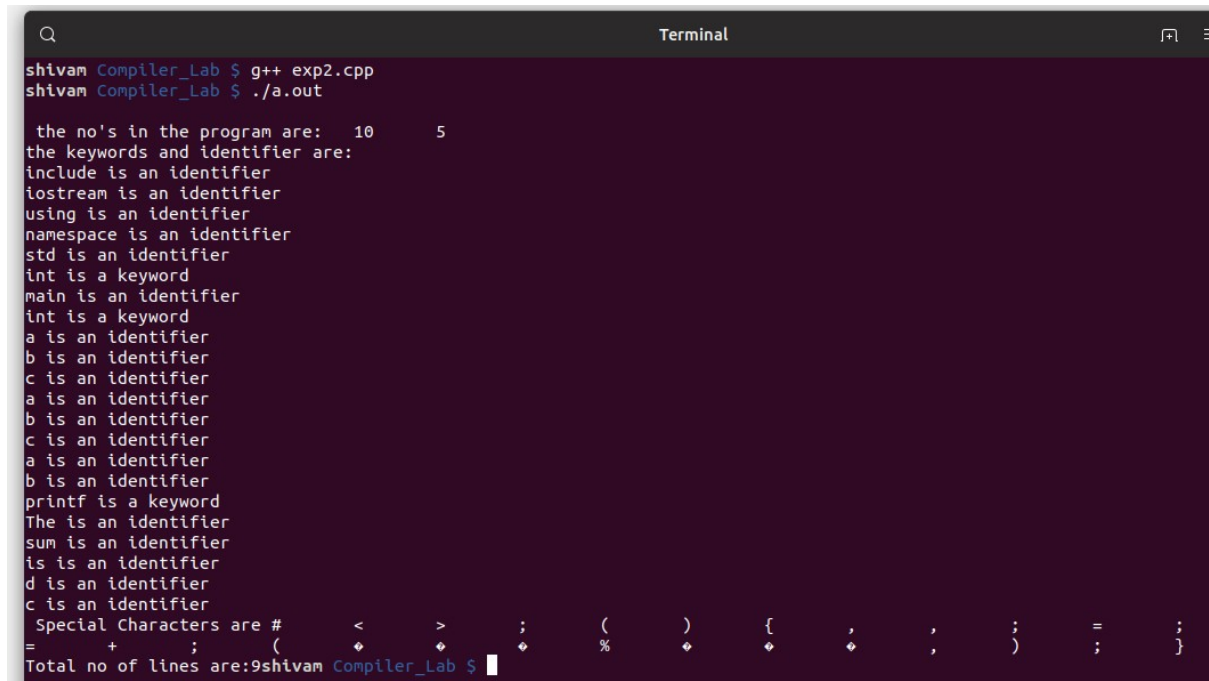
## TEST FILE:

```

#include<iostream>
using namespace std;
int main()
{
int a,b,c;
a=10;
b=5;
c=a+b;
printf("The sum is %d",c);
}

```

## OUTPUT :



```

shivam Compiler_Lab $ g++ exp2.cpp
shivam Compiler_Lab $ ./a.out

the no's in the program are: 10      5
the keywords and identifier are:
include is an identifier
iostream is an identifier
using is an identifier
namespace is an identifier
std is an identifier
int is a keyword
main is an identifier
int is a keyword
a is an identifier
b is an identifier
c is an identifier
a is an identifier
b is an identifier
c is an identifier
a is an identifier
b is an identifier
printf is a keyword
The is an identifier
sum is an identifier
is is an identifier
d is an identifier
c is an identifier
Special Characters are #      <      >      ;      (      )      {      ,      ,      ;      =      ;
=      +      ;      (      *      %      *      ,      )      ;      }
Total no of lines are:9shivam Compiler_Lab $

```

## PRACTICAL #3

### **AIM : IMPLEMENTATION OF A LEXICAL ANALYZER USING LEX TOOL.**

#### **STEPS :**

1. Lex program contains three sections: definitions, rules, and user subroutines. Each section must be separated from the others by a line containing only the delimiter, %%. The format is as follows: definitions %% rules %% user\_subroutines
2. In definition section, the variables make up the left column, and their definitions make up the right column. Any C statements should be enclosed in %{..}%. Identifier is defined such that the first letter of an identifier is alphabet and remaining letters are alphanumeric.
3. In rules section, the left column contains the pattern to be recognized in an input file to yylex(). The right column contains the C program fragment executed when that pattern is recognized. The various patterns are keywords, operators, new line character, number, string, identifier, beginning and end of block, comment statements, preprocessor directive statements etc.
4. Each pattern may have a corresponding action, that is, a fragment of C source code to execute when the pattern is matched.
5. When yylex() matches a string in the input stream, it copies the matched text to an external character array, yytext, before it executes any actions in the rules section.
6. In user subroutine section, main routine calls yylex(). yywrap() is used to get more input.
7. The lex command uses the rules and actions contained in file to generate a program, lex.yy.c, which can be compiled with the cc command. That program can then receive input, break the input into the logical pieces defined by the rules in file, and run program fragments contained in the actions in file.

#### **CODE :**

```
%{  
int COMMENT=0;  
%}  
identifier [a-zA-Z][a-zA-Z0-9]*  
%%  
#.* {printf("\n%s is a preprocessor directive",yytext);}  
int |  
float |  
char |
```



```

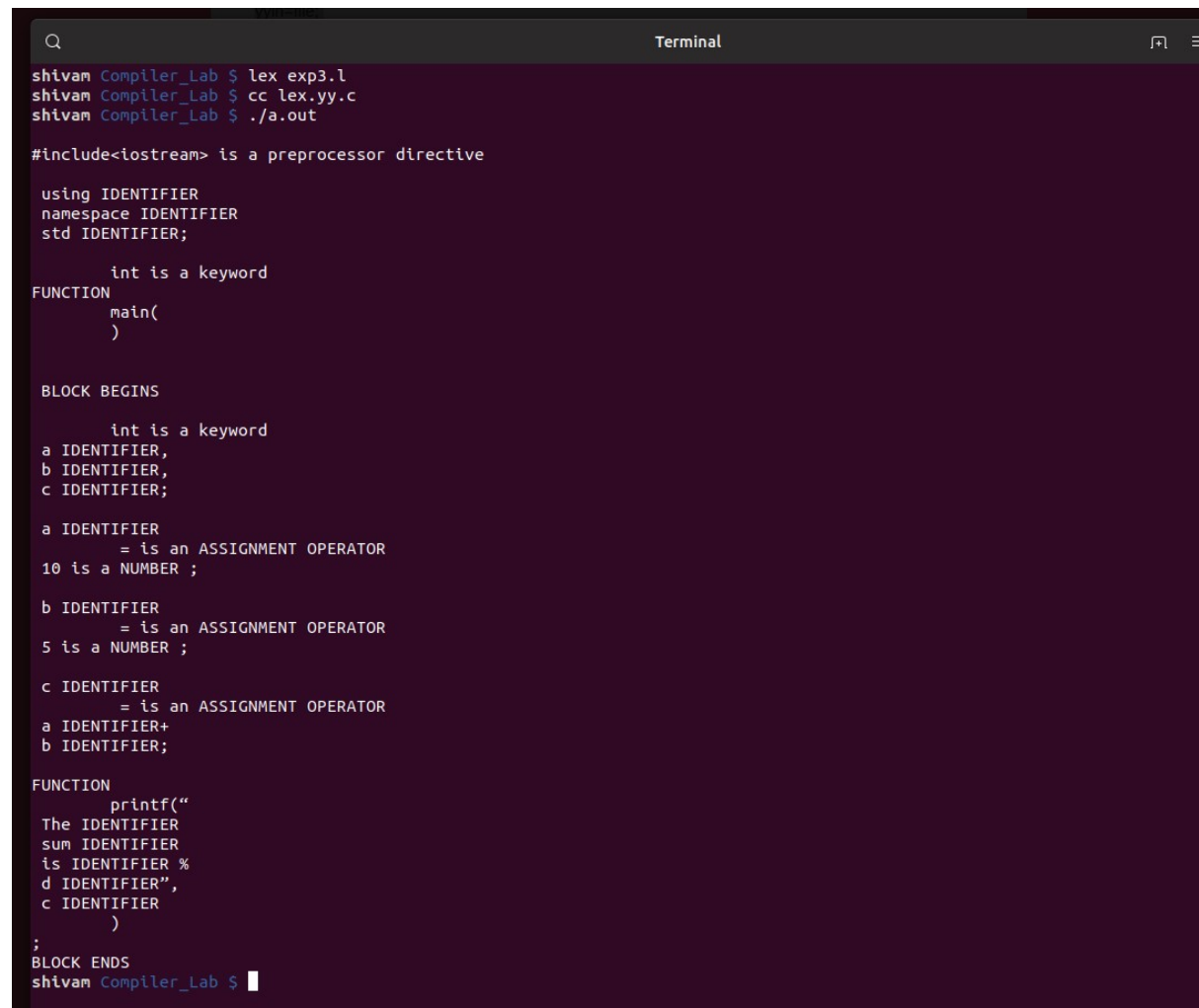
double |
while |
for |
struct |
typedef |
do |
if |
break |
continue |
void |
switch |
return |
else |
goto {printf("\n\t%s is a keyword",yytext);}
"/*" {COMMENT=1;} {printf("\n\t %s is a COMMENT",yytext);}
{identifier}\( {if(!COMMENT)printf("\nFUNCTION \n\t%s",yytext);}
\{ {if(!COMMENT)printf("\n BLOCK 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("test.cpp","r");
if(!file)
{
printf("could not open the file");
exit(0);
}
yyin=file;
yylex();
printf("\n");
return(0);
}
int yywrap()
{
return(1);
}

```

## TEST FILE:

```
#include<iostream>
using namespace std;
int main()
{
int a,b,c;
a=10;
b=5;
c=a+b;
printf("The sum is %d",c);
}
```

## OUTPUT :



```
shivam Compiler_Lab $ lex exp3.l
shivam Compiler_Lab $ cc lex.yy.c
shivam Compiler_Lab $ ./a.out

#include<iostream> is a preprocessor directive

using IDENTIFIER
namespace IDENTIFIER
std IDENTIFIER;

    int is a keyword
FUNCTION
    main(
    )

BLOCK BEGINS

    int is a keyword
a IDENTIFIER,
b IDENTIFIER,
c IDENTIFIER;

a IDENTIFIER
  = is an ASSIGNMENT OPERATOR
10 is a NUMBER ;

b IDENTIFIER
  = is an ASSIGNMENT OPERATOR
5 is a NUMBER ;

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

FUNCTION
    printf("
The IDENTIFIER
sum IDENTIFIER
is IDENTIFIER %
d IDENTIFIER",
c IDENTIFIER
    )
;
BLOCK ENDS
shivam Compiler_Lab $
```

## PRACTICAL #4A

**AIM : Generate YACC specification for a few syntactic categories.**

**a) Program to recognize a valid arithmetic expression that uses operator +, -, \*, and /.**

### **STEPS :**

1. Start the program.
2. Reading an expression .
3. Checking the validating of the given expression according to the rule using yacc.
4. Using expression rule print the result of the given values
5. Stop the program.

### **LEX CODE :**

```
%{
    #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 CODE :**

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

```

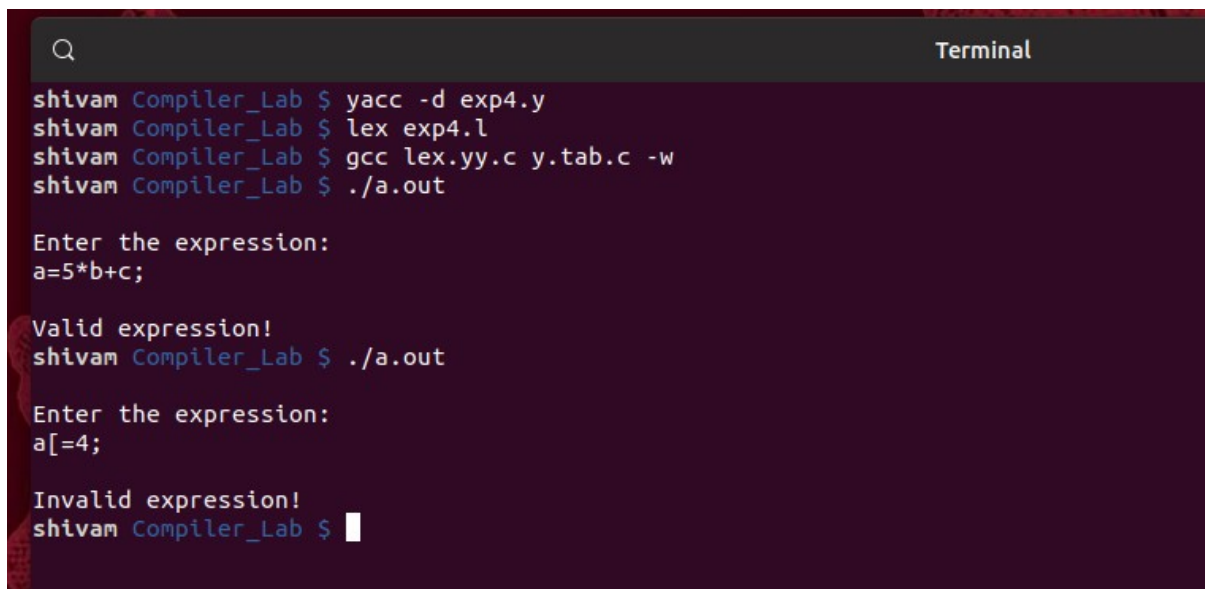
        | '(' s ')' x
        ;
x :      op s
    | '-' s
    ;
%%

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 :



```

shivam Compiler_Lab $ yacc -d exp4.y
shivam Compiler_Lab $ lex exp4.l
shivam Compiler_Lab $ gcc lex.yy.c y.tab.c -w
shivam Compiler_Lab $ ./a.out

Enter the expression:
a=5*b+c;

Valid expression!
shivam Compiler_Lab $ ./a.out

Enter the expression:
a[=4;

Invalid expression!
shivam Compiler_Lab $

```

## **PRACTICAL #4B**

**AIM : Generate YACC specification for a few syntactic categories.**

**b) Program to recognize a valid variable which starts with a letter followed by any number of letters or digits.**

### **STEPS :**

1. Start the program
2. Reading an expression
3. Checking the validating of the given expression according to the rule using yacc.
4. Using expression rule print the result of the given values
5. Stop the program

### **LEX CODE :**

```
%{
    #include "y.tab.h"
}%
%%
[a-zA-Z_][a-zA-Z_0-9]* return letter;
[0-9]          return digit;
.              return yytext[0];
\n             return 0;
%%
int yywrap()
{
    return 1;
}
```

### **YACC CODE :**

```
%{
    #include <stdio.h>
    int valid=1;
}%
%token digit letter
%%
start : letter s
s :    letter s
    | digit s
```

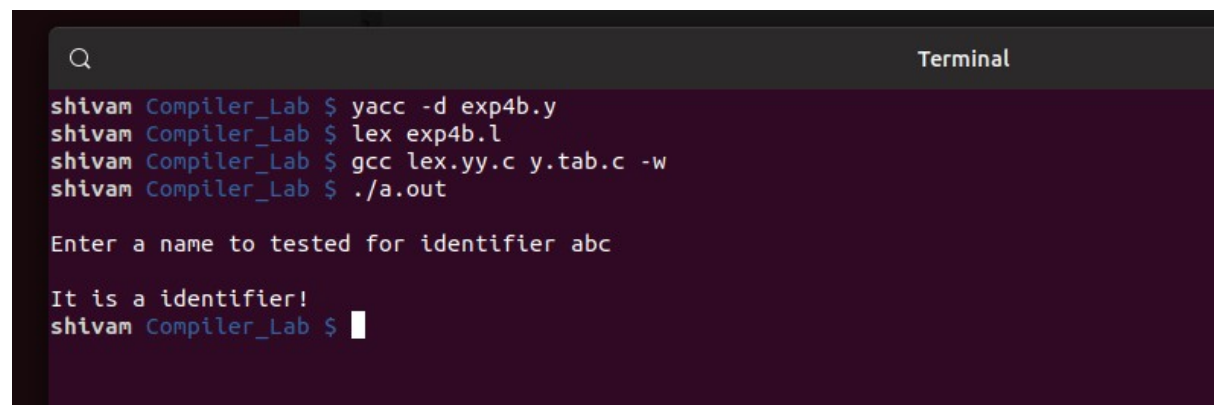
```

|
;
%%
int yyerror()
{
    printf("\nIts not a identifier!\n");
    valid=0;
    return 0;
}

int main()
{
    printf("\nEnter a name to tested for identifier ");
    yyparse();
    if(valid)
    {
        printf("\nIt is a identifier!\n");
    }
}

```

## OUTPUT :



A terminal window titled "Terminal" with a search icon in the top left. The terminal shows the following commands and output:

```

shivam Compiler_Lab $ yacc -d exp4b.y
shivam Compiler_Lab $ lex exp4b.l
shivam Compiler_Lab $ gcc lex.yy.c y.tab.c -w
shivam Compiler_Lab $ ./a.out

Enter a name to tested for identifier abc

It is a identifier!
shivam Compiler_Lab $

```

## PRACTICAL #4C

**AIM : Generate YACC specification for a few syntactic categories.**

### **c)Implementation of Calculator using LEX and YACC**

#### **STEPS :**

1. A Yacc source program has three parts as follows:  
Declarations %% translation rules %% supporting C routines
2. Declarations Section: This section contains entries that:
  - i. Include standard I/O header file.
  - ii. Define global variables.
  - iii. Define the list rule as the place to start processing.
  - iv. Define the tokens used by the parser.
  - v. Define the operators and their precedence.
3. Rules Section: The rules section defines the rules that parse the input stream. Each rule of a grammar production and the associated semantic action.
4. Programs Section: The programs section contains the following subroutines. Because these subroutines are included in this file, it is not necessary to use the yacc library when processing this file.
5. Main- The required main program that calls the yyparse subroutine to start the program.
6. yyerror(s) -This error-handling subroutine only prints a syntax error message.
7. yywrap -The wrap-up subroutine that returns a value of 1 when the end of input occurs. The calc.lex file contains include statements for standard input and output, as programmer file information if we use the -d flag with the yacc command. The y.tab.h file contains definitions for the tokens that the parser program uses.
8. calc.lex contains the rules to generate these tokens from the input stream.

#### **LEX CODE :**

```
%{
#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;
}
```

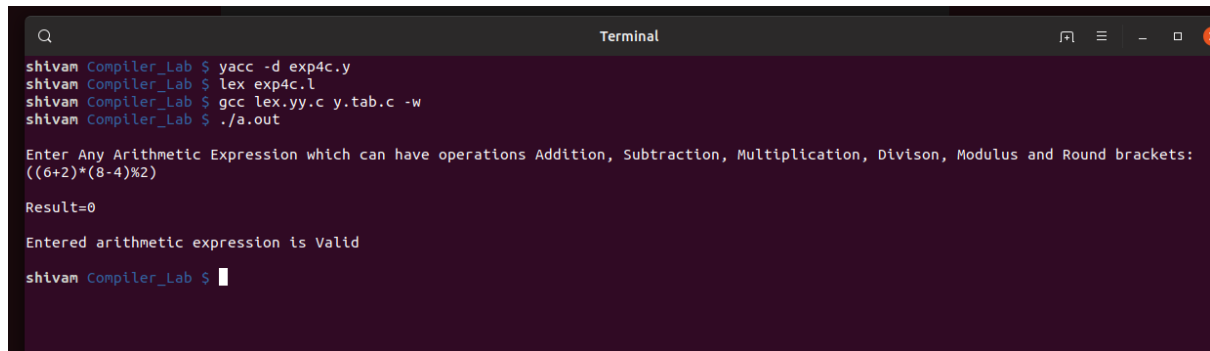
## YACC CODE :

```
%{
#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 :

A terminal window titled "Terminal" with a search icon and window controls. It shows a series of commands and their outputs. The commands are: `yacc -d exp4c.y`, `lex exp4c.l`, `gcc lex.yy.c y.tab.c -w`, and `./a.out`. The output includes a prompt for an arithmetic expression, the input `((6+2)*(8-4)%2)`, the result `Result=0`, and a validation message.

```
shivan Compiler_Lab $ yacc -d exp4c.y
shivan Compiler_Lab $ lex exp4c.l
shivan Compiler_Lab $ gcc lex.yy.c y.tab.c -w
shivan Compiler_Lab $ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:
((6+2)*(8-4)%2)

Result=0

Entered arithmetic expression is Valid

shivan Compiler_Lab $
```

## PRACTICAL #5

**AIM : Convert the BNF rules into Yacc form and write code to generate Abstract Syntax Tree.**

### **STEPS :**

1. Reading an expression.
2. Calculate the value of given expression
3. Display the value of the nodes based on the precedence.
4. Using expression rule print the result of the given value

### **LEX CODE :**

```
%{
#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 CODE :

```
%{
#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 ';'
| CONDEST
| 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
;
```

```
CONDEST: 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
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);
}

```

## TEST CODE :

```

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 :

```
shivan Compiler_Lab $ lex exp5.l
shivan Compiler_Lab $ yacc -d exp5.y
shivan Compiler_Lab $ gcc lex.yy.c y.tab.c -ll -lm -w
shivan Compiler_Lab $ ./a.out test5.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

```
shivan Compiler_Lab $
```



## PRACTICAL #6

### **AIM : Implementation of Type Checking**

#### **STEPS :**

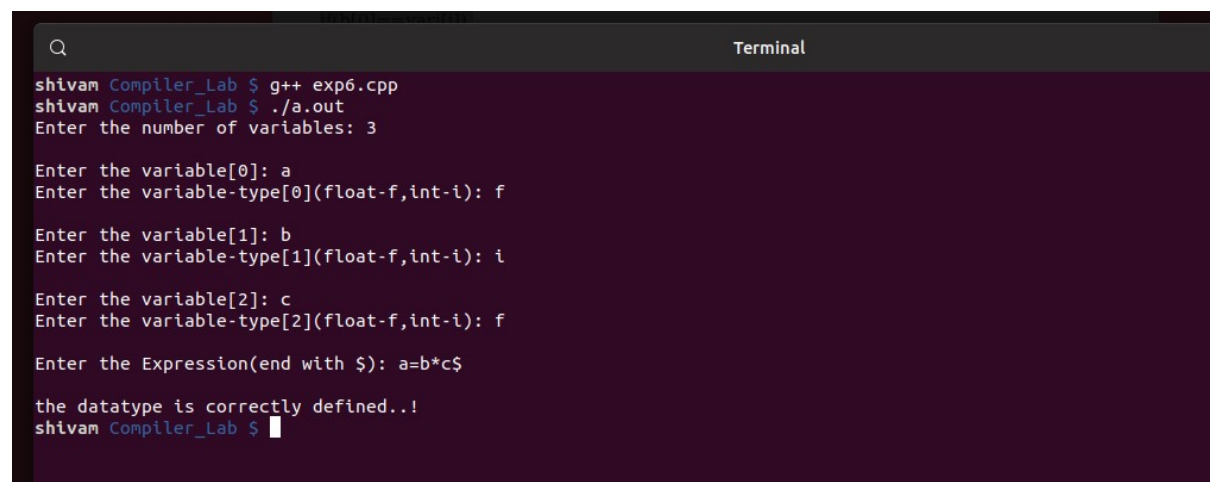
1. Track the global scope type information (e.g. classes and their members)
2. Determine the type of expressions recursively, i.e. bottom-up, passing the resulting types upwards.
3. If type found correct, do the operation
4. Type mismatches, semantic error will be notified

#### **CODE :**

```
#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("\nEnter 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("\nEnter 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 :



```
shivam Compiler_Lab $ g++ exp6.cpp
shivam Compiler_Lab $ ./a.out
Enter the number of variables: 3

Enter the variable[0]: a
Enter the variable-type[0](float-f,int-i): f

Enter the variable[1]: b
Enter the variable-type[1](float-f,int-i): i

Enter the variable[2]: c
Enter the variable-type[2](float-f,int-i): f

Enter the Expression(end with $): a=b*c$

the datatype is correctly defined..!
shivam Compiler_Lab $
```

## PRACTICAL #7

**AIM : Implement control flow analysis and data flow analysis.**

**CODE :**

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

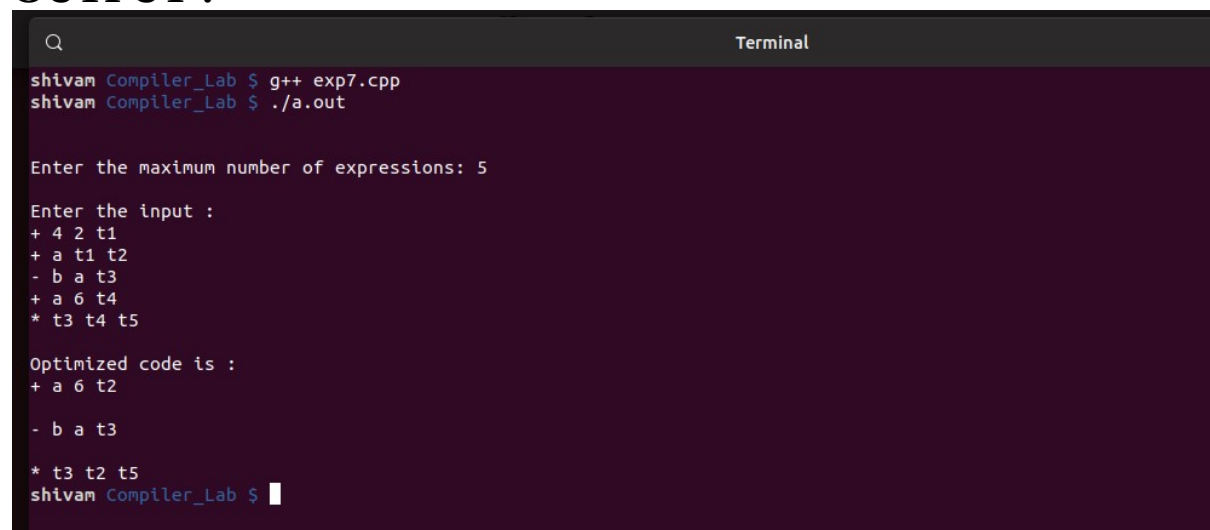
```

```

}}
}}
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 :



```

shivan@Compiler_Lab:~$ g++ exp7.cpp
shivan@Compiler_Lab:~$ ./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
shivan@Compiler_Lab:~$

```

## PRACTICAL #8

**AIM : Implement any one storage allocation strategies(heap, stack, static)**

### **STEPS :**

1. Initially check whether the stack is empty.
2. Insert an element into the stack using push operation.
3. Insert more elements onto the stack until stack becomes full.
4. Delete an element from the stack using pop operation.
5. Display the elements in the stack.
6. Top the stack element will be display.

### **CODE :**

```
#include<stdio.h>
#include<stdlib.h>
#define TRUE 1
#define FALSE 0
typedef struct Heap
{
int data;
struct Heap *next;
}
node;
node *create();
int 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;
node *get_node();
temp=NULL;
flag=TRUE;
do
{
printf("\nEnter 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("Do you want to enter more elements?(y/n): ");
scanf("%s",&ans);
}
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("\nEnter 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 :

```
Terminal
shivam Compiler_Lab $ g++ exp8.cpp
shivam Compiler_Lab $ ./a.out

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): 1

Enter the element: 45
Do you want to enter more elements?(y/n): y

Enter the element: 34
Do you want to enter more elements?(y/n): y

Enter the element: 22
Do you want to enter more elements?(y/n): n

The list is created

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
45->34->22->NULL
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
shivam Compiler_Lab $
```

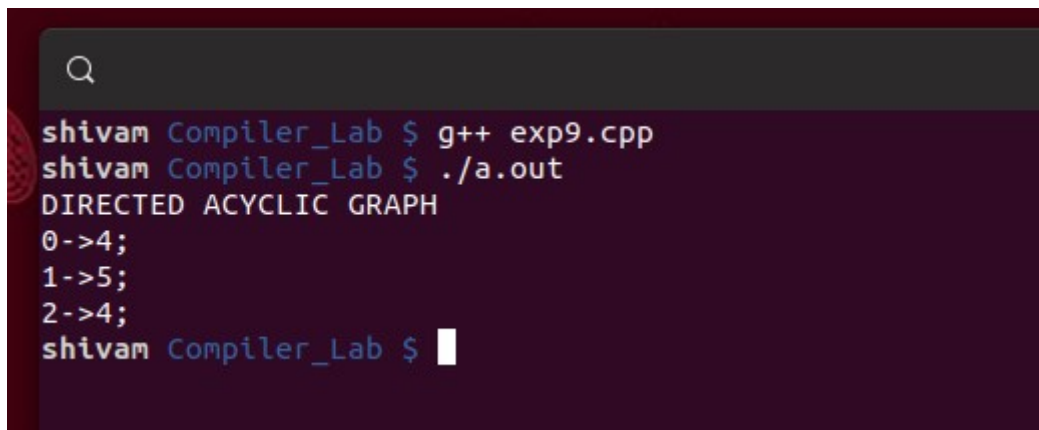
## PRACTICAL #9

### AIM : Construction of DAG

### CODE :

```
#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
int 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 :



```
shivam Compiler_Lab $ g++ exp9.cpp
shivam Compiler_Lab $ ./a.out
DIRECTED ACYCLIC GRAPH
0->4;
1->5;
2->4;
shivam Compiler_Lab $
```

## PRACTICAL #10

**AIM : Implement the back end of the compiler which takes the three address code and produces the 8086 assembly language instruction**

### **STEPS :**

1. Start the program.
2. Get the three variables from statements and stored in the text file k.txt.
3. Compile the program and give the path of the source file.
4. Execute the program.
5. Target code for the given statement was produced.
6. Stop the program.

### **CODE :**

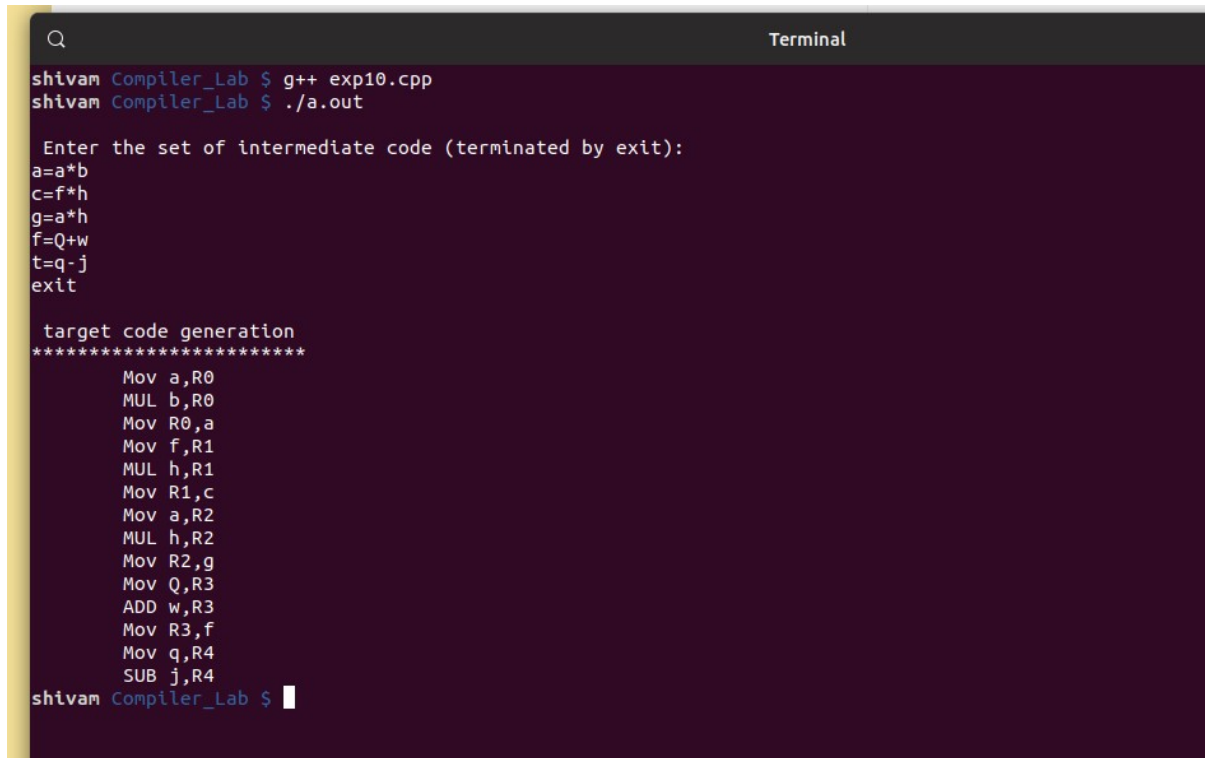
```
#include<stdio.h>
#include<string.h>
int main() {
    char icode[10][30], str[20], opr[10];
    int i = 0;
    printf("\nEnter the set of intermediate code (terminated by exit):\n");
    do
    {
        scanf("%s", icode[i]);
    } while (strcmp(icode[i++], "exit") != 0);
    printf("\n target code 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 '/':
                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);
}

```

## OUTPUT :



```

shivam Compiler_Lab $ g++ exp10.cpp
shivam Compiler_Lab $ ./a.out

Enter the set of intermediate code (terminated by exit):
a=a*b
c=f*h
g=a*h
f=Q+w
t=q-j
exit

target code generation
*****
    Mov a,R0
    MUL b,R0
    Mov R0,a
    Mov f,R1
    MUL h,R1
    Mov R1,c
    Mov a,R2
    MUL h,R2
    Mov R2,g
    Mov Q,R3
    ADD w,R3
    Mov R3,f
    Mov q,R4
    SUB j,R4
shivam Compiler_Lab $

```