



**KALASALINGAM**  
**ACADEMY OF RESEARCH AND EDUCATION**  
**(DEEMED TO BE UNIVERSITY)**



Under sec. 3 of UGC Act 1956. Accredited by NAAC with "A++" Grade

Anand Nagar, Krishnankoli, Srivilliputtur (Via), Virudhunagar (Dt) - 626126, Tamil Nadu | [info@kalasalingam.ac.in](mailto:info@kalasalingam.ac.in) | [www.kalasalingam.ac.in](http://www.kalasalingam.ac.in)

**SCHOOL OF COMPUTING**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**

**LAB MANUAL**

**213INT2302 Principles of compiler Design Lab**

**Student Name :** \_\_\_\_\_

**Register Number :** \_\_\_\_\_

**Year :** \_\_\_\_\_

**Semester :** \_\_\_\_\_

**Branch :** \_\_\_\_\_



**KALASALINGAM**  
**ACADEMY OF RESEARCH AND EDUCATION**  
**(DEEMED TO BE UNIVERSITY)**

Under sec. 3 of UGC Act 1956. Accredited by NAAC with "A++" Grade



Anand Nagar, Krishnankoll, Srivilliputtur (Via), Virudhunagar (Dt) - 626126, Tamil Nadu | [info@kalasalingam.ac.in](mailto:info@kalasalingam.ac.in) | [www.kalasalingam.ac.in](http://www.kalasalingam.ac.in)

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**BONAFIDE CERTIFICATE**

This is a bonafide record of work done by ..... studying  
in ..... Year/Semester in the .....laboratory  
during the Even semester of the academic year 2025-2026.

Signature of the Faculty-in -Charge

Signature of Head of Department

Submitted for the practical examination held at Kalasalingam Academy of Research and  
Education , Anand Nagar, Krishnankovil on .....

Register No:

--	--	--	--	--	--	--	--	--	--

Internal Examiner

External Examiner

UNIVERSITY VISION	UNIVERSITY MISSION
To be a University of Excellence of International Repute in Education and Research	<p>M1: To provide a scholarly teaching- learning ambience which results in creating graduates equipped with skills and acumen to solve real-life problems.</p> <p>M2: To promote research and create knowledge for human welfare, rural and societal development. M3: To nurture entrepreneurial ambition, industrial and societal connect by creating an environment through which innovators and leaders emerge</p>

INFORMATION TECHNOLOGY DEPARTMENT	
VISION	MISSION
To be a department of repute offering programmes in frontier areas of IT through quality education, research and imbibing societal values.	<ul style="list-style-type: none"> <li>To provide quality education through effective curriculum and innovative teaching.</li> <li>To facilitate conducive learning environment for students and faculty to investigate knowledge.</li> <li>To instill the ethical behavior and social responsibilities to provide sustainable information technology solutions</li> </ul>

PROGRAMME EDUCATIONAL OBJECTIVES
<p>Within a few years of obtaining an undergraduate degree in Computer Science and Engineering, the students will be able to:</p> <p><b>PEO-1:</b> Build successful careers or undertake higher studies and research in IT-related fields.</p> <p><b>PEO-2:</b> Develop innovative products and implement novel solutions to real world problems in IT and related interdisciplinary areas</p> <p><b>PEO-3:</b> Understand the societal impact of IT services and apply their skills in an ethical, responsible and professional manners</p>

PROGRAMME SPECIFIC OUTCOMES
<p>At the end of the Programme, the students will be able to:</p> <p><b>PSO-1:</b> Ability to identify, design and develop processes and systems for enterprises</p> <p><b>PSO-2:</b> Ability to identify, deploy and maintain the IT infrastructure based on the needs of the businesses.</p> <p><b>PSO-3:</b> Practice and promote information technologies for societal needs</p>

## **PROGRAMME OUTCOMES**

- PO1: Engineering Knowledge:** Apply knowledge of mathematics, natural science, computing, engineering fundamentals and an engineering specialization as specified in WK1 to WK4 respectively to develop to the solution of complex engineering problems.
- PO2: Problem Analysis:** Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions with consideration for sustainable development. (WK1 to WK4).
- PO3: Design/Development of Solutions:** Design creative solutions for complex engineering problems and design/develop systems/components/processes to meet identified needs with consideration for the public health and safety, whole-life cost, net zero carbon, culture, society and environment as required. (WK5).
- PO4: Conduct Investigations of Complex Problems:** Conduct investigations of complex engineering problems using research-based knowledge including design of experiments, modelling, analysis & interpretation of data to provide valid conclusions. (WK8).
- PO5: Engineering Tool Usage:** Create, select and apply appropriate techniques, resources and modern engineering & IT tools, including prediction and modelling recognizing their limitations to solve complex engineering problems. (WK2 and WK6).
- PO6: The Engineer and The World:** Analyze and evaluate societal and environmental aspects while solving complex engineering problems for its impact on sustainability with reference to economy, health, safety, legal framework, culture and environment. (WK1, WK5, and WK7).
- PO7: Ethics:** Apply ethical principles and commit to professional ethics, human values, diversity and inclusion; adhere to national & international laws. (WK9).
- PO8: Individual and Collaborative Team Work:** Function effectively as an individual, and as a member or leader in diverse/multi-disciplinary teams.
- PO9: Communication:** Communicate effectively and inclusively within the engineering community and society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations considering cultural, language, and learning differences.
- PO10: Project Management and Finance:** Apply knowledge and understanding of engineering management principles and economic decision-making and apply these to one's own work, as a member and leader in a team, and to manage projects and in multidisciplinary environments.
- PO11: Life-Long Learning:** Recognize the need for, and have the preparation and ability for i) independent and life-long learning ii) adaptability to new and emerging technologies and iii) critical thinking in the broadest context of technological change. (WK8).

**Mapping of the Programs's Student Outcomes (PSOs) With the ABET  
Student Outcomes (ASOs)**

<b>ASOs</b>	<b>ABET-CAC Student outcomes (ASOs)/CO</b>
<b>ASO1.</b>	Analyze a complex computing problem and to apply principles of computing and other relevant disciplines to identify solutions.
<b>ASO2.</b>	Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline.
<b>ASO3.</b>	Communicate effectively in a variety of professional contexts.
<b>ASO4.</b>	Recognize professional responsibilities and make informed judgments in computing practice based on legal and ethical principles
<b>ASO5.</b>	Function effectively as a member or leader of a team engaged in activities appropriate to the program's discipline.
<b>ASO6.</b>	Identify and analyze user needs to take them into account in the selection, creation, integration, evaluation, and administration of computing-based systems.

## **SYLLABUS**

### **COMPILER LABORATORY**

#### **OBJECTIVES:**

The student should be made to:

- ✓ Be exposed to compiler writing tools.
- ✓ Learn to implement the different Phases of compiler
- ✓ Be familiar with control flow and data flow analysis
- ✓ Learn simple optimization techniques

#### **LIST OF EXPERIMENTS:**

1. Implementation of Symbol Table
2. Develop a lexical analyzer to recognize a few patterns in C. (Ex. identifiers, constants, comments, operators etc.)
3. Implementation of Lexical Analyzer using Lex Tool
4.
  - 4.a Program to recognize a valid arithmetic expression that uses operator +, -, \*, and /.
  - 4.b Program to recognize a valid variable which starts with a letter followed by any number of letters or digits.
5. Implementation of Calculator using LEX and YACC
6. Implement type checking
7. Convert the BNF rules into Yacc form and write code to generate Abstract Syntax Tree.
8. Construction of DAG
9. Implement the back end of the compiler which takes the three address code and produces the 8086 assembly language instructions that can be assembled and run using a 8086 assembler. The target assembly instructions can be simple move, add, sub, jump. Also simple addressing modes are used.
10. Implementation of Simple Code Optimization Techniques (Constant Folding. etc.)

## TABLE OF CONTENTS

EXP /NO.	Date	Experiment Name	Marks	Staff Sign
1.		Implementation of Symbol Table		
2.		Develop a lexical analyzer to recognize a few patterns in C. (Ex. identifiers, constants, comments, operators etc.)		
3.		Implementation of Lexical Analyzer using Lex Tool		
4.		Recognition of Expression		
4.a.		Program to recognize a valid arithmetic expression that uses operator +, - , * and /.		
4.b.		Program to recognize a valid variable which starts with a letter followed by any number of letters or digits.		
5.		Implementation of Calculator using LEX and YACC		
6.		Implement type checking		
7.		Convert the BNF rules into Yacc form and write code to generate Abstract Syntax Tree.		
8.		Construction of DAG		
9.		Implement the back end of the compiler which takes the three address code and produces the 8086 assembly language instructions that can be assembled and run using a 8086 assembler. The target assembly instructions can be simple move, add, sub, jump. Also simple addressing modes are used.		
10.		Implementation of simple code optimization Techniques (Constant Folding. etc.)		

**Ex.No : 1**

**DATE:**

### **IMPLEMENTATION OF SYMBOL TABLE**

**AIM:**

To write a C program to implement Symbol Table.

**ALGORITHM:**

1. Start the program for performing insert, display, delete, search and modify option in symbol table
2. Define the structure of the Symbol Table
3. Enter the choice for performing the operations in the symbol Table
4. If the entered choice is 1, search the symbol table for the symbol to be inserted. If the symbol is already present, it displays "Duplicate Symbol". Else, insert the symbol and the corresponding address in the symbol table.
5. If the entered choice is 2, the symbols present in the symbol table are displayed.
6. If the entered choice is 3, the symbol to be deleted is searched in the symbol table. If it is not found in the symbol table it displays "Label Not found". Else, the symbol is deleted.
7. If the entered choice is 5, the symbol to be modified is searched in the symbol table. The label or address or both can be modified.



## **PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
struct table
{
    char var[10];
    int value;
};
struct table tbl[20];
int i,j,n;
void create();
void modify();
int search(char variable[],int n);
void insert();
void display();
void main()
{
    int ch,result=0;
    char v[10];
    clrscr();
    do
    {
        printf("Enter your choice\n1.Create\n2.Insert\n3.Modify\n4.Search\n5.Display\n6.Exit:");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
                create();
                break;
            case 2:
                insert();
                break;
            case 3:
                modify();
                break;
            case 4:
                printf("Enter the variable to be searched for:\n");
                scanf("%s",&v);
                result=search(v,n);
                if(result==0)
                    printf("The variable does not belong to the table\n");
                else
```

```

        printf("The location of the variable is %d\nThe value of %s is
%d\n",result,tbl[result].var,tbl[result].value);
        break;
    case 5:
        display();
        break;
    case 6:
        exit(1);
    }
}
while(ch!=6);
getch();
}
void create()
{
    printf("Enter the no. of entries:");
    scanf("%d",&n);
    printf("Enter the variable and the value:\n");
    for(i=1;i<=n;i++)
    {
        scanf("%s%d",tbl[i].var,&tbl[i].value);
        check:
        if(tbl[i].var[0]>='0'&&tbl[i].var[0]<='9')
        {
            printf("The variable should start with an alphabet\nEnter correct variable name:\n");
            scanf("%s%d",tbl[i].var,&tbl[i].value);
            goto check;
        }
        check1:
        for(j=1;j<i;j++)
        {
            if(strcmp(tbl[i].var,tbl[j].var)==0)
            {
                printf("The variable already exists.Enter another variable\n");
                scanf("%s%d",tbl[i].var,&tbl[i].value);
                goto check1;
            }
        }
    }
    printf("The table after creation is:\n");
    display();
}
void insert()
{
    if(i>=20)
        printf("Cannot insert.Table is full\n");
    else
    {

```

```

n++;
printf("Enter the value and variable\n");
scanf("%s%d",tbl[n].var,&tbl[n].value);
check:
if(tbl[i].var[0]>='0'&&tbl[i].var[0]<='9')
{
    printf("The variable should start with an alphabet\nEnter correct variable name:\n");
    scanf("%s%d",tbl[i].var,&tbl[i].value);
    goto check;
}
check1:
for(j=1;j<n;j++)
{
    if(strcmp(tbl[j].var,tbl[i].var)==0)
    {
        printf("The variable already exist.Enter another variable\n");
        scanf("%s%d",tbl[i].var,&tbl[i].value);
        goto check1;
    }
}
printf("The table after insertion is:\n");
display();
}
}
void modify()
{
    char variable[10];
    int result=0;
    printf("Enter the variable to be modified\n");
    scanf("%s",&variable);
    result=search(variable,n);
    if(result==0)
        printf("%s does not belong to table\n",variable);
    else
    {
        printf("The current value of the variable %s is %d\nEnter the new variable and its value\n",tbl[result].var,tbl[result].value);
        scanf("%s%d",tbl[result].var,&tbl[result].value);
        check:
        if(tbl[i].var[0]>='0'&&tbl[i].var[0]<='9')
        {
            printf("The variable should start with an alphabet\nEnter correct variable name:\n");
            scanf("%s%d",tbl[i].var,&tbl[i].value);
            goto check;
        }
    }
    printf("The table after modification is:\n");
    display();
}

```

```

}
int search(char variable[],int n)
{
    int flag;
    for(i=1;i<=n;i++)
    {
        if(strcmp(tbl[i].var,variable)==0)
        {
            flag=1;
            break;
        }
    }
    if(flag==1)
        return i;
    else
        return 0;
}
void display()
{
    printf("VARIABLE\t VALUE\n");
    for(i=1;i<=n;i++)
        printf("%s\t\t%d\n",tbl[i].var,tbl[i].value);
}

```

## **OUTPUT**

Enter your choice

- 1.Create
- 2.Insert
- 3.Modify
- 4.Search
- 5.Display
- 6.Exit:1

Enter the no. of entries:3

Enter the variable and the value:

AIM	45
ASK	34
BALL	56

The table after creation is:

VARIABLE	VALUE
AIM	45
ASK	34
BALL	56

Enter your choice

- 1.Create
- 2.Insert
- 3.Modify
- 4.Search
- 5.Display
- 6.Exit:2

Enter the value and variable

SIM	25
-----	----

The table after insertion is:

VARIABLE	VALUE
AIM	45
ASK	34
BALL	56
SIM	25

Enter your choice

- 1.Create
- 2.Insert
- 3.Modify
- 4.Search
- 5.Display
- 6.Exit:3

Enter the variable to be modified

ASK

The current value of the variable ASK is 34

Enter the new variable and its value

RIM                    40

The table after modification is:

VARIABLE	VALUE
AIM	45
RIM	40
BALL	56
SIM	25

Enter your choice

- 1.Create
- 2.Insert
- 3.Modify
- 4.Search
- 5.Display
- 6.Exit:4

Enter the variable to be searched for:

RIM

The location of the variable is 2

The value of RIM is 40

Enter your choice

- 1.Create
- 2.Insert
- 3.Modify
- 4.Search
- 5.Display
- 6.Exit:5

VARIABLE	VALUE
AIM	45
RIM	40
BALL	56
SIM	25

Enter your choice

- 1.Create
- 2.Insert
- 3.Modify
- 4.Search
- 5.Display
- 6.Exit:6

### **RESULT:**

Thus the C program to implement Symbol Table was executed and verified successfully.

**Ex.No :2**

**DATE:**

**DEVELOPMENT OF LEXICAL ANALYSER**

**AIM:**

To develop a lexical analyzer to recognize a few patterns in C.

**ALGORITHM:**

1. Open the input file in read mode.
2. Read the string from file till end of file.
3. If the character is a slash then call skipcomment() for skip the comment statements in the file.
4. If the first string matches with any delimiters operator then print as a delimiter.
5. If the first string matches with any header then print that string as a header file.
6. If the first string matches with any operator then print that as an operator.
7. If the first string matches with any keywords then print that string as a keyword.
8. If the string is not a keyword then print that as an identifier.

## **PROGRAM:**

```
/* This program is for creating a Lexical Analyzer.          lexical.c */
#include<stdio.h>
#include<string.h>                                     //Necessary Header files
#include<conio.h>
#include<ctype.h>
FILE *fp;                                             //File pointer for accessing the file
char delim[14]={ ' ','\t','\n',';',';',';',';','(' , ')','{','}','[' , ']', '#','<','>','};
char oper[7]={ '+','-', '*', '/', '%', '=', '!','}; //Array holding all the operators

/* Set of Keywords to check and list out */
char key[21][12]= { "int", "float", "char", "double", "bool", "void", "extern", "unsigned", "goto",
"static", "class", "struct", "for", "if", "else", "return", "register", "long", "while", "do"};

/*The Preprocessor directives*/
char predirect[2][12]={ "include", "define"};

/*The possible header files*/
char header[6][15]={ "stdio.h", "conio.h", "malloc.h", "process.h", "string.h", "ctype.h"};
void skipcomment();    //Function to skip comments.
void analyze();        //Function analyzing the input file
void check(char[]);    //Function to check whether the string matches any of the keywords
int isdelim(char);     //Function to check if the character retrieved from the file is a delimiter
int isop(char);        //Function to check if the character retrieved from the file is an operator
int fop=0,numflag=0,f=0;    //necessary flag values
char c,ch,sop;           //necessary character variables to store the characters retrieved

void main()                //Main Function
{
    char fname[12];
    clrscr();
    printf("\nEnter filename : ");
    scanf("%s",fname);
    fp=fopen(fname,"r");    //Opening the file
    if(fp==NULL)           //Checking Existence of the File
        printf("\nThe file doesn't exist.");
```



```

else                                //If Check succeeds

    analyze();

printf("\nEnd of file\n");
getch();

}                                    //End of Main function


void analyze()                      //Analyse Function
{
    char token[50]; //Declare a token character array
    int j=0;
    while(!feof(fp)) //While the file is not over
    {
        c=getc(fp);
        if(c=='/')           //checking for comments in the file
        {
            skipcomment(); //Function to skip comment statements
        }
        else if(c=="")        //Skipping printf statements and other such display
            while((c=getc(fp))!="");
        else if(isalpha(c))    //checking if the character is an alphabet or not
        {
            if(numflag==1)
            {
                token[j]='\0';
                check(token);
                numflag=0;
                j=0;
                f=0;
            }
            else
            {
                token[j]=c;    //Combining the characters to get the token
                j++;
            }
            if(f==0)
                f=1;
        }
    }
}

```

```

else if(isalnum(c))
{
    if(numflag==0)
        numflag=1;
    token[j]=c;//combining the characters to get the token
    j++;
}
else
{
    if(isdelim(c))                //Checking for delimiters.
    {
        if(numflag==1)
        {
            token[j]='\0';
            check(token);
            numflag=0;
        }
        if(f==1)
        {
            token[j]='\0';
            numflag=0;
            check(token);
        }
        j=0;
        f=0;
        printf("\nDelimiter\t %c",c);
    }
    else if(isop(c))            //Checking for operators
    {
        if(numflag==1)
        {
            token[j]='\0';
            check(token);
            numflag=0;
            j=0;
            f=0;
        }
    }
}

```

```

        if(f==1)
        {
            token[j]='\0';
            j=0;
            f=0;
            numflag=0;
            check(token);
        }
    if(fop==1)
    {
        fop=0;
        printf("\nOperator\t %c%c",c,sop); //In case the operator is like '++' or '--'
    }
    else
        printf("\nOperator\t %c",c); //In other cases.
}
else if(c=='.'')
{
    token[j]=c;
    j++;
}
}
}
}

```

int isdelim(char c) //Function to check if the character retrieved from the file is a delimiter.

```

{
    int i;
    for(i=0;i<14;i++)
    {
        if(c==delim[i])
            return 1;
    }
    return 0;
}

```

int isop(char c) //Function to check if the character retrieved from the file is an operator.

```

{
    int i,j;
    char ch;
    for(i=0;i<7;i++)
    {
        if(c==oper[i])
        {
            ch=getc(fp);
            for(j=0;j<6;j++)          //In case the operator is like '++' or '--', etc.
            {
                if(ch==oper[j])
                {
                    fop=1;
                    sop=ch;
                    return 1;
                }
            }
            ungetc(ch,fp);
            return 1;
        }
    }
    return 0;
}

```

void check(char t[]) //Function to check if the token is an identifier, keyword, header file name

```

{
    int i;
    if(numflag==1)
    {
        printf("\nNumber\t\t%s",t);
        return;
    }
    for(i=0;i<2;i++)
    {
        if(strcmp(t,predirect[i])==0)
        {
            printf("\nPreprocessor directive %s",t);
            return;
        }
    }
}

```

```

        }
    }
    for(i=0;i<6;i++)
    {
        if(strcmp(t,header[i])==0)
        {
            printf("\nHeader file\t%s",t);
            return;
        }
    }
    for(i=0;i<21;i++)
    {
        if(strcmp(key[i],t)==0)
        {
            printf("\nKeyword\t\t%s",key[i]);
            return;
        }
    }
    printf("\nIdentifier\t%s",t);
}

void skipcomment()    //Function to skip over the comment statements in the file.
{
    ch=getc(fp);
    if(ch=='/')    //Checking single line comments
    {
        while((ch=getc(fp))!='\n');
    }
    else if(ch=='*') //Checking multiple line comments.
    {
        while(f==0)
        {
            ch=getc(fp);
            if(ch=='*')
            {
                c=getc(fp);
                if(c=='/')

```

```

        f=1;
    }
}
f=0;
}
}

```

### INPUT FILE: iplex.c

```

#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
printf("Welcome\n");
getch();
}

```

### OUTPUT

Enter filename : iplex.c

Delimiter    #  
Preprocessor directive include  
Delimiter    <  
Header file    stdio.h  
Delimiter    >  
Delimiter

Delimiter    #  
Preprocessor directive include  
Delimiter    <  
Header file    conio.h  
Delimiter    >  
Delimiter

Keyword       void  
Delimiter  
Identifier    main

Delimiter (

Delimiter )

Delimiter

Delimiter {

Delimiter

Identifier clrscr

Delimiter (

Delimiter )

Delimiter ;

Delimiter

Identifier printf

Delimiter (

Delimiter )

Delimiter ;

Delimiter

Identifier getch

Delimiter (

Delimiter )

Delimiter ;

Delimiter

Delimiter }

End of file

## **RESULT**

Thus the C program for implementation of a lexical analyzer to recognize a few patterns was executed and verified successfully.

**Ex. No: 3**

**DATE:**

### **IMPLEMENTATION OF LEXICAL ANALYSER USING LEX TOOL**

**AIM:**

To write a 'C' program to implement a lexical analyzer for separation of tokens using LEX Tool.

**ALGORITHM:**

**Step 1:** Declare and Initialize the required variable.

**Step 2:** If the statement starts with #.\* print it as preprocessor directive.

**Step 3:** Check for the given list of keywords and print them as keyword if it is encountered.

**Step 4:** If the given string is '/\*' or '\*/' print it as comment line.

**Step 5:** For a function, print the beginning and ending of the function block.

**Step 6:** Similarly print the corresponding statements for numbers, identifiers and assignment operators.

**Step 7:** In the main function get the input file as argument and open the file in read mode.

**Step 8:** Then read the file and print the corresponding lex statement given above.

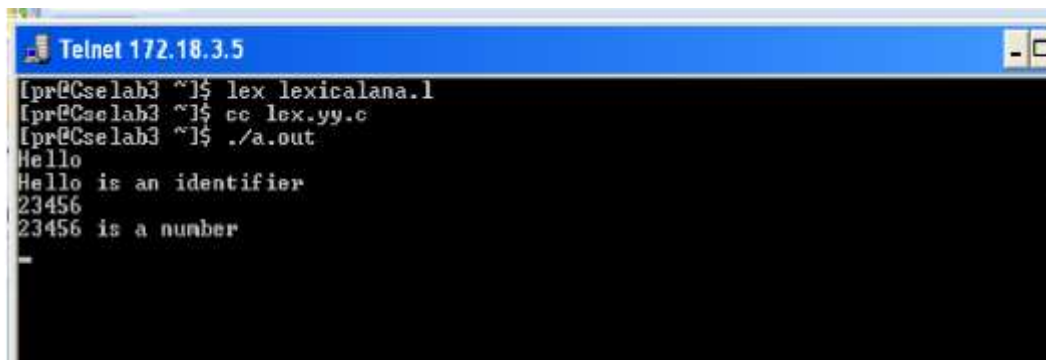


## **PROGRAM 1:**

### **Program Name: id.l**

```
% {  
    #include<stdio.h>  
% }  
  
%%  
if|else|while|int|switch|for    { printf("%s is a keyword",yytext);}  
[a-zA-Z]([a-zA-Z][a-zA-Z0-9])*  { printf("%s is an identifier",yytext);}  
[0-9]*                          { printf("%s is a number",yytext);}  
%%  
int main()  
{  
    yylex();  
    return 0;  
}  
int yywrap()  
{  
}
```

## **OUTPUT**



The screenshot shows a Telnet window titled "Telnet 172.18.3.5". The user is logged in as "pr@cselab3". The following commands and outputs are shown:

```
[pr@cselab3 ~]$ lex lexicalana.l  
[pr@cselab3 ~]$ cc lex.yy.c  
[pr@cselab3 ~]$ ./a.out  
Hello  
Hello is an identifier  
23456  
23456 is a number  
-
```

## **PROGRAM 2:**

```
% {
% }
identifier [a-z|A-Z][a-z|A-Z|0-9]*

%%
#.*          { printf("\n%s is a preprocessor dir",yytext);}
int          { printf("\n\t%s is a keyword",yytext);}
{identifier}\( { printf("\n\nFUNCTION\n\t%s",yytext);}
\{          { printf("\nBLOCK BEGINS");}
\}          { printf("\nBLOCK ENDS");}
{identifier} { printf("\n%s is an IDENTIFIER",yytext);}
. | \n

%%

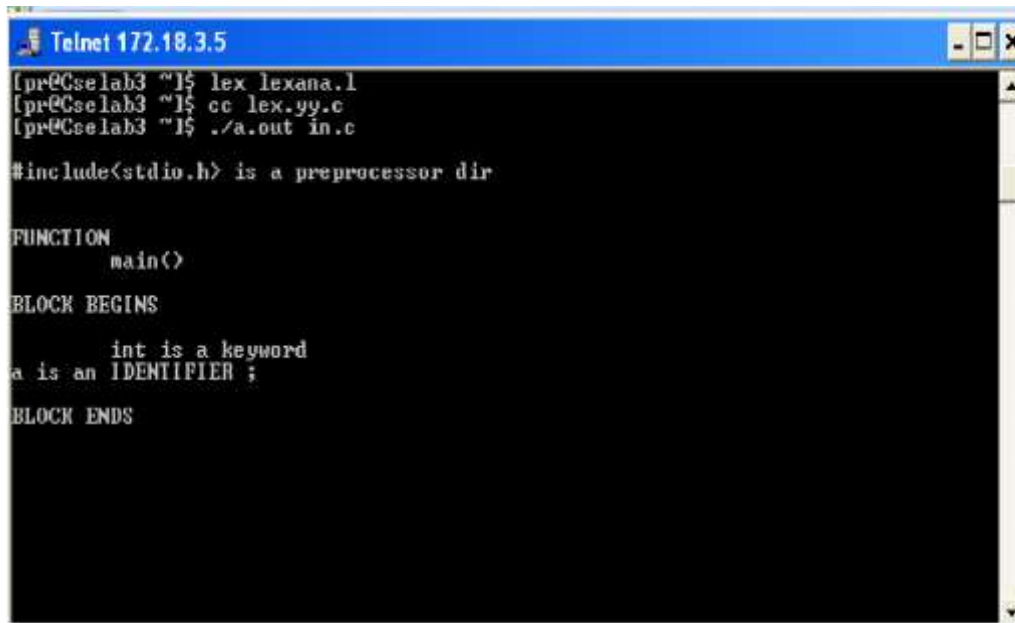
int main(int argc,char **argv)
{
    if(argc>1)
    {
        FILE *file;
        file=fopen(argv[1],"r");
        if(!file)
        {
            printf("\n couldnot open %s\n",argv[1]);
            exit(0);
        }
        yyin=file;
    }
    yylex();
    printf("\n\n");
    return 0;
}

int yywrap()
{
    return 0;
}
```

## **Input ( in.c )**

```
#include<stdio.h>
main()
{
    int a ;
}
```

## OUTPUT:



```
Telnet 172.18.3.5
[pr@cse1ab3 ~]$ lex lexana.l
[pr@cse1ab3 ~]$ cc lex.yy.c
[pr@cse1ab3 ~]$ ./a.out in.c

#include<stdio.h> is a preprocessor dir

FUNCTION
    main()
BLOCK BEGINS
    int is a keyword
a is an IDENTIFIER ;
BLOCK ENDS
```

## RESULT:

Thus the C program for the implementation of lexical analyzer using LEX Tool was executed successfully.

**Ex. No: 4 (a)**

**DATE:**

**GENERATION OF YACC SPECIFICATION  
RECOGNIZING A VALID ARITHMETIC EXPRESSION**

**AIM:**

To write a program to recognize a valid arithmetic expression that uses operator +, -, \* and / using YACC tool.

**ALGORITHM:**

**LEX**

1. Declare the required header file and variable declaration with in '%{' and '%}'.
2. LEX requires regular expressions to identify valid arithmetic expression token of lexemes.
3. LEX call **yywrap()** function after input is over. It should return 1 when work is done or should return 0 when more processing is required.

**YACC**

1. Declare the required header file and variable declaration with in '%{' and '%}'.
2. Define tokens in the first section and also define the **associativity** of the operations
3. Mention the grammar productions and the action for each production.
4. **\$\$** refer to the top of the stack position while **\$1** for the first value, **\$2** for the second value in the stack.
5. Call **yyparse()** to initiate the parsing process.
6. **yyerror()** function is called when all productions in the grammar in second section doesn't match to the input statement.

## **PROGRAM:**

**//art\_expr.l**

```
% {  
    #include<stdio.h>  
    #include "y.tab.h"  
  
% }  
%%  
    [a-zA-Z][0-9a-zA-Z]* {return ID;}  
    [0-9]+ {return DIG;}  
    [ \t]+ {;}  
    . {return yytext[0];}  
    \n {return 0;}  
%%  
int yywrap()  
{  
    return 1;  
}
```

**//art\_expr.y**

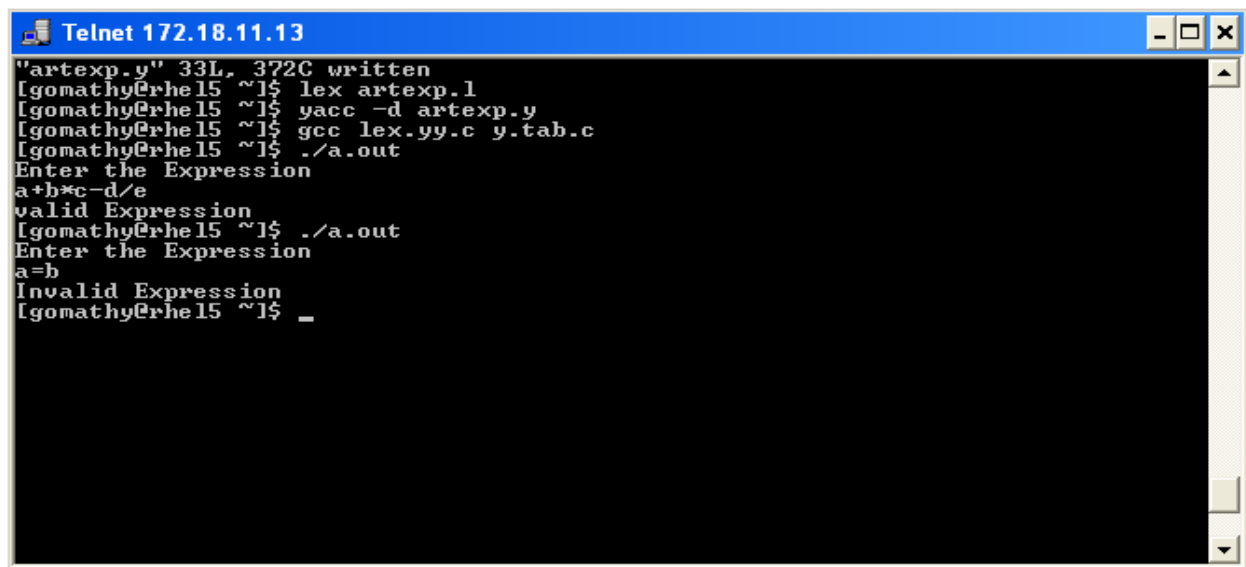
```
% {  
    #include<stdio.h>  
  
% }  
%token ID DIG  
%left '+' '-'  
%left '*' '/'  
%right UMINUS  
%%  
    stmt:expn ;  
    expn:expn '+' expn  
        |expn '-' expn  
        |expn '*' expn  
        |expn '/' expn  
        | '-' expn %prec UMINUS  
        | '(' expn ')'  
        | DIG  
        | ID
```

```

;
%%
int main()
{
    printf("Enter the Expression \n");
    yyparse();
    printf("valid Expression \n");
    return 0;
}
int yyerror()
{
    printf("Invalid Expression");
    exit(0);
}

```

## OUTPUT



```

Telnet 172.18.11.13
"artexp.y" 33L, 372C written
lgomathy@rhe15 ~1$ lex artexp.l
lgomathy@rhe15 ~1$ yacc -d artexp.y
lgomathy@rhe15 ~1$ gcc lex.yy.c y.tab.c
lgomathy@rhe15 ~1$ ./a.out
Enter the Expression
a+b*c-d/e
valid Expression
lgomathy@rhe15 ~1$ ./a.out
Enter the Expression
a=b
Invalid Expression
lgomathy@rhe15 ~1$ _

```

## RESULT:

Thus the program to recognize a valid arithmetic expression that uses operator +, -, \*, and / using YACC tool was executed and verified successfully.

**Ex. No: 4 (b)**

**DATE:**

## **RECOGNIZING A VALID VARIABLE**

### **AIM:**

To write a program to recognize a valid variable which starts with a letter followed by any number of letters or digits using YACC tool.

### **ALGORITHM:**

#### **LEX**

1. Declare the required header file and variable declaration with in ‘%{‘ and ‘%}’.
2. LEX requires regular expressions or patterns to identify token of lexemes for recognize a valid variable.
3. Lex call **yywrap()** function after input is over. It should return 1 when work is done or should return 0 when more processing is required.

#### **YACC**

1. Declare the required header file and variable declaration with in ‘%{‘ and ‘%}’.
2. Define tokens in the first section and also define the **associativity** of the operations
3. Mention the grammar productions and the action for each production.
4. **\$\$** refer to the top of the stack position while **\$1** for the first value, **\$2** for the second value in the stack.
5. Call **yparse()** to initiate the parsing process.
6. **yyerror()** function is called when all productions in the grammar in second section doesn't match to the input statement.

## **PROGRAM:**

**//valvar.l**

```
% {  
    #include "y.tab.h"  
% }  
%%  
    [a-zA-Z]    { return LET; }  
    [0-9]        { return DIG; }  
    .            { return yytext[0]; }  
    \n          { return 0; }  
%%  
int yywrap()  
{  
    return 1;  
}
```

**//valvar.y**

```
% {  
    #include <stdio.h>  
% }  
%token LET DIG  
%%  
    variable:var  
    ;  
    var:var DIG  
    |var LET  
    |LET  
    ;  
%%  
int main()  
{  
    printf("Enter the variable:\n");  
    yyparse();  
    printf("Valid variable \n");  
}
```

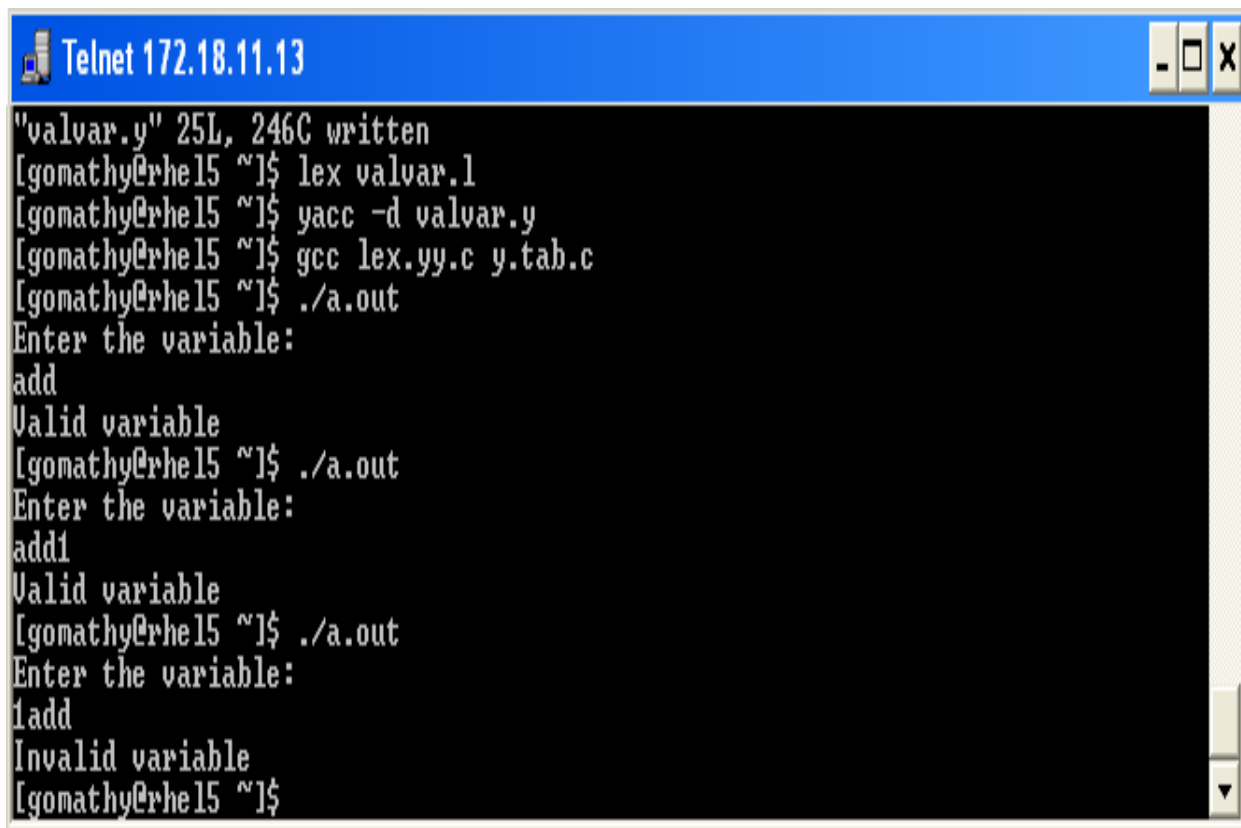


```

        return 0;
    }
    int yyerror()
    {
        printf("Invalid variable \n");
        exit(0);
    }
}

```

### **OUTPUT:**



```

Telnet 172.18.11.13
"valvar.y" 25L, 246C written
[gomathy@rhe15 ~]$ lex valvar.l
[gomathy@rhe15 ~]$ yacc -d valvar.y
[gomathy@rhe15 ~]$ gcc lex.yy.c y.tab.c
[gomathy@rhe15 ~]$ ./a.out
Enter the variable:
add
Valid variable
[gomathy@rhe15 ~]$ ./a.out
Enter the variable:
add1
Valid variable
[gomathy@rhe15 ~]$ ./a.out
Enter the variable:
1add
Invalid variable
[gomathy@rhe15 ~]$

```

### **RESULT:**

Thus the program to recognize a valid variable which starts with a letter followed by any number of letters or digits using YACC tool was executed and verified successfully.

**Ex.NO: 4(c)**

**DATE:**

### **IMPLEMENTATION OF CALCULATOR USING LEX AND YACC**

**AIM:**

To write a program to implement Calculator using LEX and YACC.

**ALGORITHM:**

**Step 1:** Start the program.

**Step 2:** In the declaration part of lex, includes declaration of regular definitions as digit.

**Step 3:** In the translation rules part of lex, specifies the pattern and its action that is to be executed whenever a lexeme matched by pattern is found in the input in the call.

**Step 4:** By use of Yacc program, all the Arithmetic operations are done such as +, -, \*, /.

**Step 5:** Display error is persist.

**Step 6:** Provide the input.

**Step 7:** Verify the output.

**Step 8:** End.

## **PROGRAM:**

**cal.l**

DIGIT [0-9]+

%option noyywrap

%%

```
{DIGIT}      { yylval=atof(yytext);  return NUM;}
\n|.         { return yytext[0];}
```

%%

**cal.y**

% {

#include<ctype.h>

#include<stdio.h>

#define YYSTYPE double

% }

%token NUM

%left '+' '-'

%left '\*' '/'

%right UMINUS

%%

Statment:E { printf("Answer: %g \n", \$\$); }

|Statment '\n'

;

E : E+'E { \$\$ = \$1 + \$3; }

| E-'E { \$\$=\$1-\$3; }

| E'\*E { \$\$=\$1\*\$3; }

| E'/E { \$\$=\$1/\$3; }

| NUM

;

%%

**OUTPUT:**

```
"cal2.y" 59L, 1186C written  
[exam01@Cselab3 ~]$ lex cal2.l  
[exam01@Cselab3 ~]$ yacc yaccal2.y  
[exam01@Cselab3 ~]$ cc y.tab.c  
[exam01@Cselab3 ~]$ ./a.out  
Enter the expression:2+2  
Answer: 4
```

**RESULT:**

Thus the program for implementing calculator using LEX and YACC is executed and verified.

**Ex.No:5**

**DATE:**

**CONVERSION OF THE BNF RULES INTO YACC FORM AND GENERATION ABSTRACT SYNTAX  
TREE**

**AIM:**

To write the program to convert the BNF rules into YACC form and write code to generate abstract syntax tree.

**ALGORITHM:**

1. Start the program.
2. Reading an input file line by line.
3. Convert it in to abstract syntax tree using three address codes.
4. Represent three address codes in the form of quadruple tabular form.

## **PROGRAM:**

**<int.l>**

```
% {
    #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];
```

%%

**<int.y>**

```
% {
    #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 Arg1 Arg2 Result" "\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);
}

```

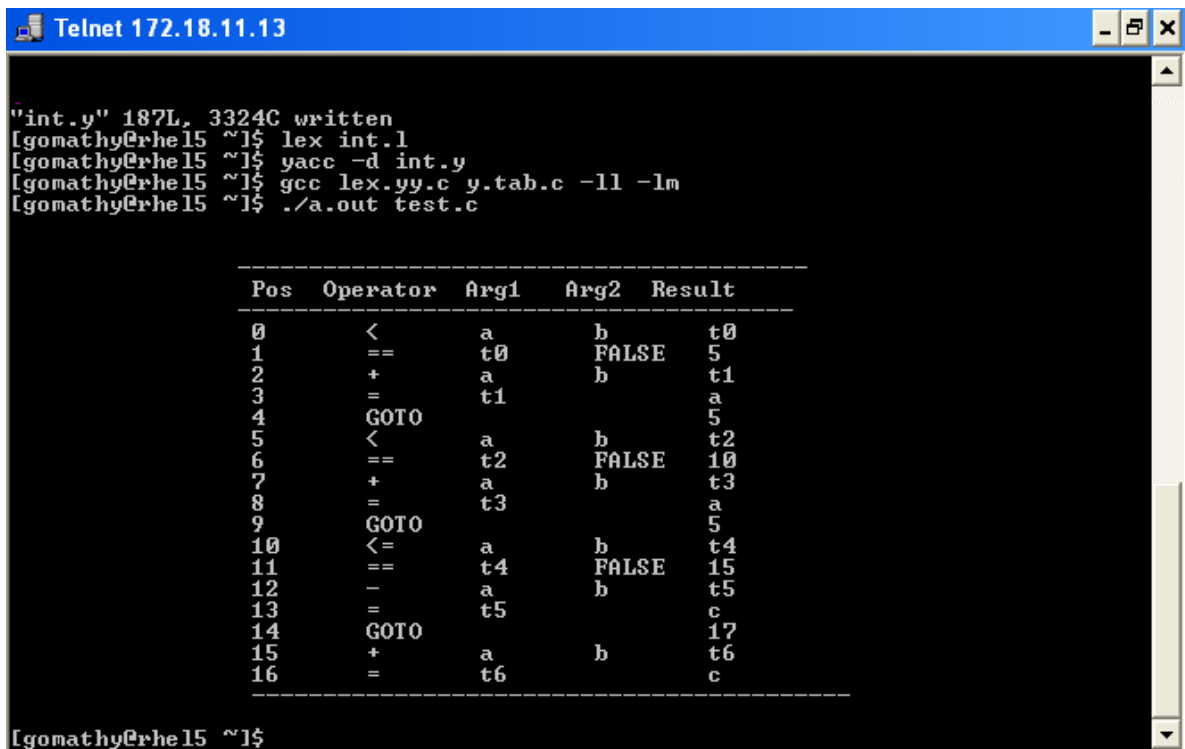
```
yyerror()
{
    printf("\n Error on line no:%d",LineNo);
}
```

### **Input**

\$vi test.c

```
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:



```
"int.y" 187L, 3324C written
[gomath@rhe15 ~]$ lex int.l
[gomath@rhe15 ~]$ yacc -d int.y
[gomath@rhe15 ~]$ gcc lex.yy.c y.tab.c -ll -lm
[gomath@rhe15 ~]$ ./a.out test.c

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

-----
[gomath@rhe15 ~]$
```

## RESULT:

Thus the program to convert the BNF rules into YACC forms and writes code to generate abstract syntax tree was executed successfully.

**Ex.No: 6**

**DATE:**

### **IMPLEMENTATION OF TYPE CHECKING**

**AIM:**

To write a C program to implement type checking.

**ALGORITHM:**

1. Start the program for type checking of given expression
2. Read the expression and declaration
3. Based on the declaration part define the symbol table
4. Check whether the symbols present in the symbol table or not. If it is found in the symbol table it displays "Label already defined".
5. Read the data type of the operand 1, operand 2 and result in the symbol table.
6. If the both the operands' type are matched then check for result variable. Else, print "Type mismatch".
7. If all the data type are matched then displays "No type mismatch".

## **PROGRAM:**

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
int count=1,i=0,j=0,l=0,findval=0,k=0,kflag=0;
char key[4][12]= {"int","float","char","double"};
char dstr[100][100],estr[100][100];
char token[100],resultvardt[100],arg1dt[100],arg2dt[100];
void entry();
int check(char[]);
int search(char[]);
void typecheck();

struct table
{
char var[10];
char dt[10];
};
struct table tbl[20];

void main()
{
clrscr();
printf("\n IMPLEMENTATION OF TYPE CHECKING \n");

printf("\n DECLARATION \n\n");
do
{
printf("\t");
gets(dstr[i]);
i++;
} while(strcmp(dstr[i-1],"END"));
printf("\n EXPRESSION \n\n");
do
{
printf("\t");
gets(estr[l]);
l++;
}while(strcmp(estr[l-1],"END"));

i=0;
printf("\n SEMANTIC ANALYZER(TYPE CHECKING): \n");
while(strcmp(dstr[i],"END"))
{
entry();
printf("\n");
i++;
}
l=0;
while(strcmp(estr[l],"END"))
{
typecheck();
printf("\n");
```

```

        l++;
    }

    printf("\n PRESS ENTER TO EXIT FROM TYPE CHECKING\n");
    getch();
}
void entry()
{
    j=0;
    k=0;
    memset(token,0,sizeof(token));
    while(dstr[i][j]!=' ')
    {
        token[k]=dstr[i][j];
        k++;
        j++;
    }
    kflag=check(token);
    if(kflag==1)
    {
        strcpy(tbl[count].dt,token);
        k=0;
        memset(token,0,strlen(token));
        j++;
        while(dstr[i][j]!=';')
        {
            token[k]=dstr[i][j];
            k++;
            j++;
        }
        findval=search(token);
        if(findval==0)
        {
            strcpy(tbl[count].var,token);
        }
        else
        {
            printf("The variable %s is already declared",token);
        }
        kflag=0;
        count++;
    }
    else
    {
        printf("Enter valid datatype\n");
    }
}

void typecheck()
{
    memset(token,0,strlen(token));
    j=0;
    k=0;
    while(estr[l][j]!='=')

```

```

{
    token[k]=estr[l][j];
    k++;
    j++;
}
findval=search(token);
if(findval>0)
{
    strcpy(resultvardt,tbl[findval].dt);
    findval=0;
}
else
{
    printf("Undefined Variable\n");
}
k=0;
memset(token,0,strlen(token));
j++;
while(((estr[l][j]!='+')&&(estr[l][j]!='-')&&(estr[l][j]!='*')&&(estr[l][j]!='/')))
{
    token[k]=estr[l][j];
    k++;
    j++;
}
findval=search(token);
if(findval>0)
{
    strcpy(arg1dt,tbl[findval].dt);
    findval=0;
}
else
{
    printf("Undefined Variable\n");
}
k=0;
memset(token,0,strlen(token));
j++;
while(estr[l][j]!=';')
{
    token[k]=estr[l][j];
    k++;
    j++;
}
findval=search(token);
if(findval>0)
{
    strcpy(arg2dt,tbl[findval].dt);
    findval=0;
}
else
{
    printf("Undefined Variable\n");
}
if(!strcmp(arg1dt,arg2dt))
{

```

```

        if(!strcmp(resultvardt,arg1dt))
        {
            printf("\tThere is no type mismatch in the expression %s ",estr[1]);
        }
        else
        {
            printf("\tLvalue and Rvalue should be same\n");
        }
    }
    else
    {
        printf("\tType Mismatch\n");
    }
}

```

```

int search(char variable[])
{
    int i;
    for(i=1;i<=count;i++)
    {
        if(strcmp(tbl[i].var,variable) == 0)
        {
            return i;
        }
    }
    return 0;
}

```

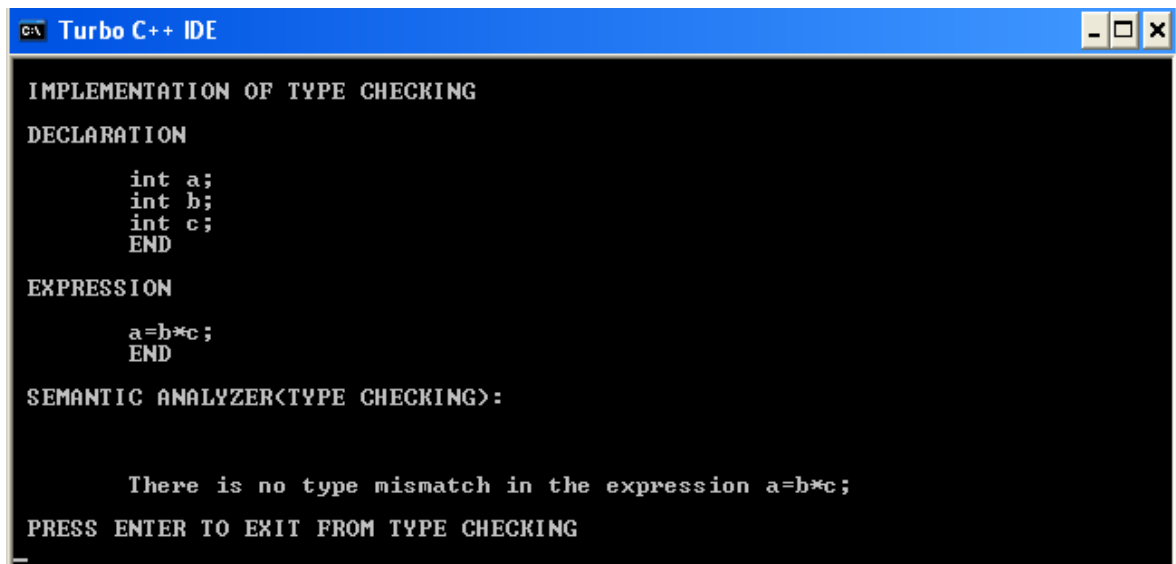
```

int check(char t[])
{
    int in;
    for(in=0;in<4;in++)
    {
        if(strcmp(key[in],t)==0)
        {
            return 1;
        }
    }
    return 0;
}

```



### OUTPUT:



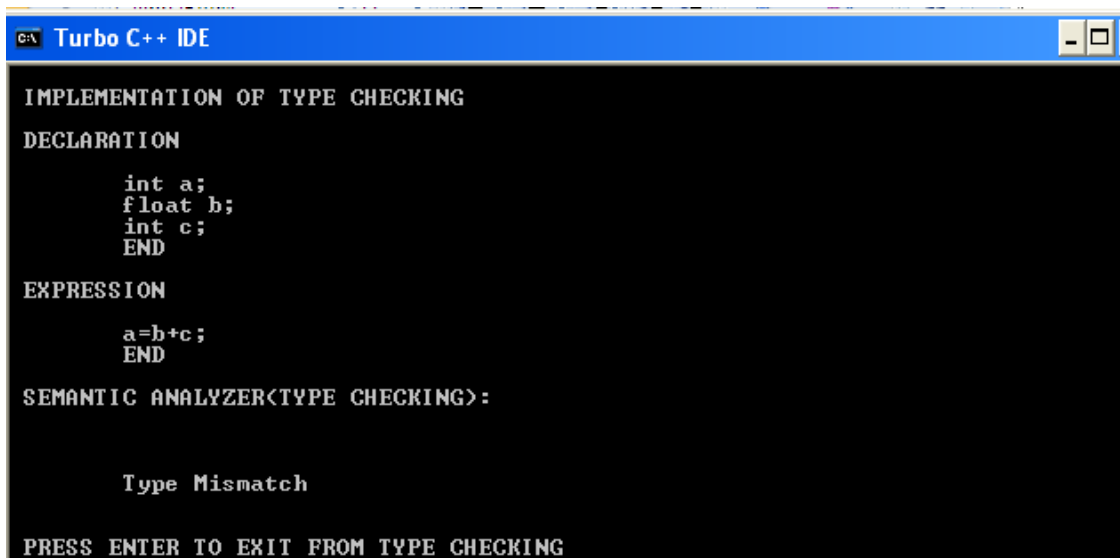
```
C:\ Turbo C++ IDE

IMPLEMENTATION OF TYPE CHECKING
DECLARATION
    int a;
    int b;
    int c;
    END

EXPRESSION
    a=b*c;
    END

SEMANTIC ANALYZER<TYPE CHECKING>:

    There is no type mismatch in the expression a=b*c;
PRESS ENTER TO EXIT FROM TYPE CHECKING
```



```
C:\ Turbo C++ IDE

IMPLEMENTATION OF TYPE CHECKING
DECLARATION
    int a;
    float b;
    int c;
    END

EXPRESSION
    a=b+c;
    END

SEMANTIC ANALYZER<TYPE CHECKING>:

    Type Mismatch
PRESS ENTER TO EXIT FROM TYPE CHECKING
```

### RESULT:

Thus the program for type checking is executed and verified.

**Ex.No: 7**

**DATE:**

## **CONSTRUCTION OF DAG**

### **AIM:**

To write a C program to construct DAG.

### **ALGORITHM:**

- 1) Read the intermediate code as input from the file.
- 2) Store the argument1 as left node and argument2 as right node.
- 3) Attach the result variable to the root node of an expression.
- 4) Use the same node if the variable and values are same.
- 5) Finally construct the DAG

## **PROGRAM:**

```
#include<stdio.h>

#include <conio.h>

struct node

{

    struct node *next;

    char id[10];

    char left[10];

    char right[10];

    char attach[3][10];

};

struct node *head;

FILE *f;

int i,s=0;

char str[25],store[10][25];

void construct_tree()

{

    struct node *temp;

    struct node *t;

    struct node *ptr;

    int flag=0,f1=0;

    temp=head;

    if(s==5||s==6)

    {

        while (temp->next!=NULL)

        {

            if(!strcmp(store[2],temp->next->left))
```

```

        flag+=1;

        if(!strcmp(store[4],temp->next->right))

            flag+=2;

        if(flag!=0)

            break;

        temp=temp->next;
    }

    t=head;

    while(t->next!=NULL)

        t=t->next;

    if(flag==0)

    {

        ptr=(struct node*)malloc(sizeof(struct node));

        t->next=ptr;

        if(s==5)

            strcpy(ptr->id,store[3]);

        else

            strcpy(ptr->id, strcat(store[3],store[5]));

        t=head;

        while(t->next!=NULL)

        {

            if(!strcmp(t->next->attach[0],store[2]))

            {

                f1=1;

                break;

            }

            if(strcmp(t->next->attach[1],""))

                if(!strcmp(t->next->attach[1],store[2]))

```

```

        {
            f1=1;
            break;
        }
        t=t->next;
    }
    if(f1)
        strcpy(ptr->left,t->next->id);
    else
        strcpy(ptr->left,store[2]);
    f1=0;
    t=head;
    while(t->next!=NULL)
    {
        if(!strcmp(t->next->attach[0],store[4]))
        {
            f1=1;
            break;
        }
        if(strcmp(t->next->attach[1],""))
        if(!strcmp(t->next->attach[1],store[4]))
        {
            f1=1;
            break;
        }
        t=t->next;
    }
    if(f1)

```

```

        strcpy(ptr->right,t->next->id);

    else

        strcpy(ptr->right,store[0]);

        strcpy(ptr->attach[1],"");

        ptr->next=NULL;

    }

    else if(flag==3)

        strcpy(temp->next->attach[1],store[0]);

    }

    if(s==3)

    {

        while(temp->next!=NULL)

        {

            if(!strcmp(store[2],temp->next->attach[0]))

                break;

            temp=temp->next;

        }

        strcpy(temp->next->attach[1],store[0]);

    }

}

int main()

{

    struct node *temp;

    struct node *t;

    clrscr();

    f=fopen("C:\\\\DAG.txt","r");

    head=(struct node*)malloc(sizeof(struct node));

```

```

head->next=NULL;

while(!feof(f))
{
    fscanf(f,"%s",str);

    if(!strcmp(str,";"))
    {
        construct_tree();

        s=0;
    }
    else
        strcpy(store[s++],str);
}

printf("\n\nID\tLEFT\tRIGHT\tATTACHED IDs\n\n");

temp=head;

while(temp->next!=NULL)
{
    printf("\n\n%s\t%s\t%s\t%s",temp->next->id,
    temp->next->left,temp->next->right,temp->next->attach[0]);

    if(strcmp(temp->next->attach[1],""))
        printf("\t%s",temp->next->attach[1]);

    temp=temp->next;
}

getch();

return 0;
}

```

**INPUT:**

```
t1=4*i;  
t2=a[t1];  
t3=4*i;  
t4=b[t3];  
t5=t2*t4;  
t6=prod+t5;  
prod=t6;  
t7=i+1;  
i=t7;
```

**OUTPUT:**

ID	LEFT	RIGHT	ATTACHED	IDs
*	4	I	t1	t3
[]	a	*	t2	
[]	b	*	t4	
*	[]	[]	t5	
+	prod	*	t6	prod
+	I		t7	i

**RESULT:**

Thus the program for construction of DAG is executed and verified.



**Ex.No: 8**

**DATE:**

## **IMPLEMENTATION OF STACK**

### **AIM:**

To implement a storage allocation using stack.

### **ALGORITHM:**

- 1) Get the size of the stack.
- 2) Read the choice of operation.
- 3) If the choice is 1 push the items into the stack
- 4) To push the elements perform the following.
  - a) If the top of stack is equal to size of stack elements can't be pushed.
  - b) Read item to be pushed.
  - c) Set or increment the top by one.
  - d) Assign the item to the stack [top].
  - e) Repeat the steps until it is required.
- 5) If choice is 2 pop the items from the stack
  - a) Check for empty stack.
  - b) Decrement the top by one.
  - c) Return the popped item.
- 6) If choice is 3 display the contents of Stack.

**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]);
        }
    }

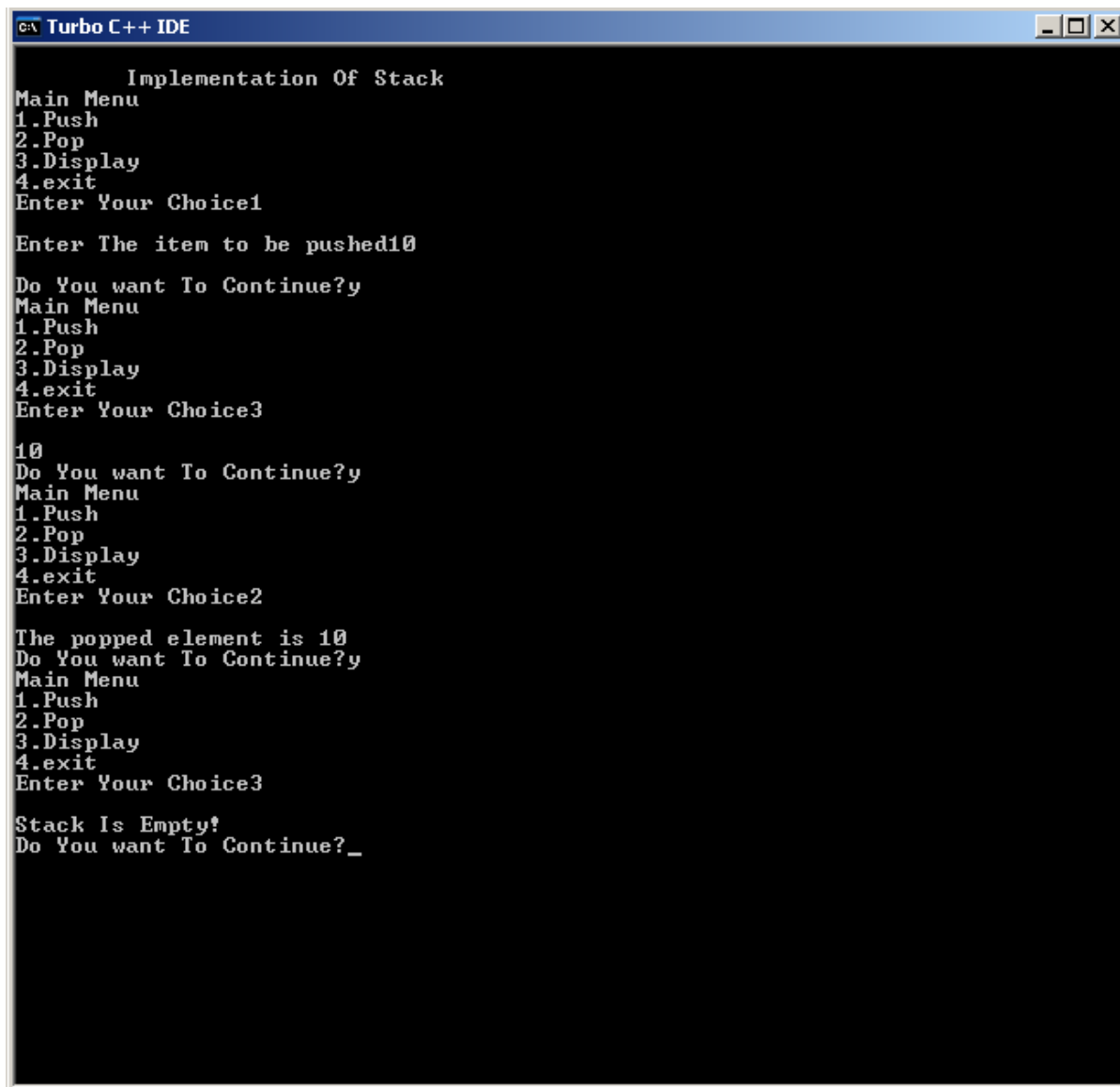
int main()
{
    int item, choice;
    char ans;
    st.top = -1;

    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("\nEmpty stack!Underflow !!");
                else
                {
                    item = pop();
                    printf("\nThe popped element is %d", item);
                }
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
        }
        printf("\nDo You want To Continue?");
        ans=getche();
    }
    while(ans=='Y'||ans =='y');
}

```

```
return 0;
}
```

### **OUTPUT:**



The screenshot shows the Turbo C++ IDE window with the following text displayed in the console:

```
Implementation Of Stack
Main Menu
1.Push
2.Pop
3.Display
4.exit
Enter Your Choice1

Enter The item to be pushed10

Do You want To Continue?y
Main Menu
1.Push
2.Pop
3.Display
4.exit
Enter Your Choice3

10
Do You want To Continue?y
Main Menu
1.Push
2.Pop
3.Display
4.exit
Enter Your Choice2

The popped element is 10
Do You want To Continue?y
Main Menu
1.Push
2.Pop
3.Display
4.exit
Enter Your Choice3

Stack Is Empty!
Do You want To Continue?_
```

### **RESULT:**

Thus the program for stack implementation is executed and verified.

**Ex.No: 9**

**DATE:**

## **IMPLEMENTATION OF BACKEND**

**AIM:**

To write a 'C' program to generate the machine code for the given intermediate code.

**ALGORITHM:**

**Step1:** Get the input expression from the user.

**Step2:** The given expression is transformed into tokens.

**Step3:** Display the assembly code according to the operators present in the given expression.

**Step4:** Use the temporary registers (R0, R1) while storing the values in assembly code programs.

## **PROGRAM:**

```
/* CODE GENERATOR */
#include<stdio.h>
#include<string.h>
int count=0,i=0,l=0;
char str[100][100];
void gen();

void main()
{
    clrscr();
    printf("\n CODE GENERATOR \n");
    printf("\n ENTER THREE ADDRESS CODE \n\n");
    do
    {
        printf("\t");
        gets(str[i]);
        i++;
    } while(strcmp(str[i-1],"QUIT"));

    i=0;
    printf("\n ASSEMBLY LANGUAGE CODE: \n");
    while(strcmp(str[i-1],"QUIT"))
    {
        gen();
        printf("\n");
        i++;
    }

    printf("\n PRESS ENTER TO EXIT FROM CODE GENERATOR\n");
    getch();
}

void gen()
{
    int j;
    printf("\n");
    for(j=strlen(str[i])-1;j>=0;j--)
    {
        char reg='R';
        if(isdigit(str[i][j])||(isalpha(str[i][j]))|| str[i][j]=='+'||str[i][j]=='-'||str[i][j]=='*'||str[i][j]=='/'||str[i][j]==' '||str[i][j]=='&'||str[i][j]=='.'||str[i][j]=='=')
        {
            switch(str[i][j])
            {
                case '+':
                    printf("\n\t MOV\t%c,%c%d",str[i][j-1],reg,count);
```

```

        printf("\n\t ADD\t%c,%c%d",str[i][j+1],reg,count);
        break;
    case '-':
        printf("\n\t MOV\t%c,%c%d",str[i][j-1],reg,count);
        printf("\n\t SUB\t%c,%c%d",str[i][j+1],reg,count);
        break;
    case '*':
        printf("\n\t MOV\t%c,%c%d",str[i][j-1],reg,count);
        printf("\n\t MUL\t%c,%c%d",str[i][j+1],reg,count);
        break;
    case '/':
        printf("\n\t MOV\t%c,%c%d",str[i][j-1],reg,count);
        printf("\n\t DIV\t%c,%c%d",str[i][j+1],reg,count);
        break;
    case '|':
        printf("\n\t MOV\t%c,%c%d",str[i][j-1],reg,count);
        printf("\n\t OR\t%c,%c%d",str[i][j+1],reg,count);
        break;
    case '&':
        printf("\n\t MOV\t%c,%c%d",str[i][j-1],reg,count);
        printf("\n\t AND\t%c,%c%d",str[i][j+1],reg,count);
        break;
    case ':':
        if(str[i][j+1]=='=')
        {
            printf("\n\t MOV\t%c,%c",reg,count,str[i][j-1]);
            count++;
        }
        else
        {
            printf("\n syntax error...\n");
        }
        break;
    default:
        break;
    }
}
else printf("\n Error\n");
}
}

```

## **OUTPUT:**

CODE GENERATOR

ENTER THREE ADDRESS CODE

A:=B+C

D:=E/F

QUIT

ASSEMBLY LANGUAGE CODE:

MOV B,R0

ADD C,R0

MOV R0,A

MOV E,R1

DIV F,R1

MOV R1,D

PRESS ENTER TO EXIT FROM CODE GENERATOR

## **RESULT:**

Thus the program for generation of Machine Code for the given intermediate code is executed and verified.



**Ex.No:10**

**DATE:**

## **IMPLEMENTATION OF CODE OPTMIZATION TECHNIQUES**

**AIM:**

To write a C program to implement Simple Code Optimization Techniques.

**ALGORITHM:**

1. Read the un-optimized input block.
2. Identify the types of optimization
3. Optimize the input block
4. Print the optimized input block
5. Execute the same with different set of un-optimized inputs and obtain the optimized input block.

## **PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
void main()
{
    char a[25][25],u,op1='*',op2='+',op3='/',op4='-';
    int p,q,r,l,o,ch,i=1,c,k,j,count=0;
    FILE *fi,*fo;
    // clrscr();
    printf("Enter three address code");
    printf("\nEnter the ctrl-z to complete:\n");
    fi=fopen("infile.txt","w");
    while((c=getchar())!=EOF)
        fputc(c,fi);
    fclose(fi);
    printf("\n Unoptimized input block\n");
    fi=fopen("infile.txt","r");
    while((c=fgetc(fi))!=EOF)
    {
        k=1;
        while(c!=';'&& c!=EOF)
        {
            a[i][k]=c;
            printf("%c",a[i][k]);
            k++;
            c=fgetc(fi);
        }
        printf("\n");
        i++;
    }
    count=i;
    fclose(fi);
    i=1;
    printf("\n Optimized three address code");
    while(i<count)
```

```

{
    if(strcmp(a[i][4],op1)==0&&strcmp(a[i][5],op1)==0)
    {
        printf("\n type 1 reduction in strength");
        if(strcmp(a[i][6],'2')==0)
        {
            for(j=1;j<=4;j++)
                printf("%c",a[i][j]);
            printf("%c",a[i][3]);
        }
    }
    else if(isdigit(a[i][3])&&isdigit(a[i][5]))
    {
        printf("\n type2 constant floding");
        p=a[i][3];
        q=a[i][5];
        if(strcmp(a[i][4],op1)==0)
            r=p*q;
        if(strcmp(a[i][4],op2)==0)
            r=p+q;
        if(strcmp(a[i][4],op3)==0)
            r=p/q;
        if(strcmp(a[i][4],op4)==0)
            r=p-q;
        for(j=1;j<=2;j++)
            printf("%c",a[i][j]);
        printf("%d",r);
        printf("\n");
    }
    else if(strcmp(a[i][5],'0')==0||strcmp(a[i][5],'1')==0)
    {
        cprintf("\n type3 algebraic expression elimation");

        if((strcmp(a[i][4],op1)==0&&strcmp(a[i][5],'1')==0)||(strcmp(a[i][4],op3)==0&&strcmp(a[i][5],'1')==0))
        {
            for(j=1;j<=3;j++)

```

```
        printf("%c",a[i][j]);  
        printf("\n");  
    }  
  
    else  
        printf("\n sorry cannot optimize\n");  
    }  
    else  
    {  
        printf("\n Error input");  
    }  
  
    i++;  
}  
getch();  
}
```

infile.txt

a=d/1; b=2+4; c=s\*\*2;

## OUTPUT

```
Enter three address code
Enter the ctrl-z to complete:
a=d/1;b=2+4;c=s**2;↵

Unoptimized input block
a=d/1
b=2+4
c=s**2

Optimized three address code

type3 algebraic expression elimination: a=d

type2 constant folding: b=6

type 1 reduction in strength: c=s*s
```

## RESULT

Thus the C program for implementation of Code optimization was executed successfully.