

KNOWLEDGE INSTITUTE OF TECHNOLOGY

(AUTONOMOUS INSTITUTION)

Approved by AICTE, New Delhi and Affiliated To Anna University,
Chennai

Kakapalayam (PO), Salem - 637 504.



RECORD NOTE BOOK

REG NO.	
---------	--

*Certified that this is the bonafide record of work done by
Selvan/Selvi.....of the.....
Semester..... Branch
during the year.....in the
..... laboratory.*

Staff-Incharge

Head of the Department

Submitted for the university Practical Examination on.....

Internal Examiner

External Examiner

Table of Contents

[illegible]

Ex. No.:1a	RECOGNIZATION OF PATTERNS IN C
Date:	

Aim:

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

Algorithm:

1. Start the program
2. Include necessary header files.
3. The ctype header file is to load the file with predicate isdigit.
4. Initialize the necessary variables.
5. Check the necessary condition.
6. Call the initialize() function. This function loads the keywords into the symbol table.
7. Check the conditions such as white spaces, digits, letters and alphanumerics.
8. To return index of entry for string S, or 0 if S is not found using lookup() function.
9. Check this until EOF is found.
10. Otherwise initialize the token value to be none.
11. In the main function if lookahead equals numeric then the value of attribute num is given by the global variable tokenval.
12. Check the necessary conditions such as arithmetic operators , parenthesis , identifiers, assignment operators and relational operators.
13. Stop the program.

Program:

Exp1.c

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

int main()
{
FILE *input, *output;
int l=1;
int t=0;
int j=0;
```

```

int i,flag;
char ch,str[20];
input = fopen("input.txt","r");
output = fopen("output.txt","w");
char keyword[30][30] = {"int","main","if","else","do","while"};
fprintf(output,"Line no. \t Token no. \t\t Token \t\t Lexeme\n\n");
while(!feof(input))
{
i=0;
flag=0;
ch=fgetc(input);
if( ch=='+' || ch=='-' || ch=='*' || ch=='/' )
{
fprintf(output,"%7d\t\t %7d\t\t Operator\t %7c\n",l,t,ch);
t++;
}
else if( ch==';' || ch=='{' || ch=='}' || ch=='(' || ch==')' || ch=='?' ||ch=='@' || ch=='!'
||ch=='%')
{
fprintf(output,"%7d\t\t %7d\t\t Special symbol\t %7c\n",l,t,ch);
t++;
}
else if(isdigit(ch))
{
fprintf(output,"%7d\t\t %7d\t\t Digit\t\t %7c\n",l,t,ch);
t++;
}
else if(isalpha(ch))
{
str[i]=ch;
i++;
ch=fgetc(input);

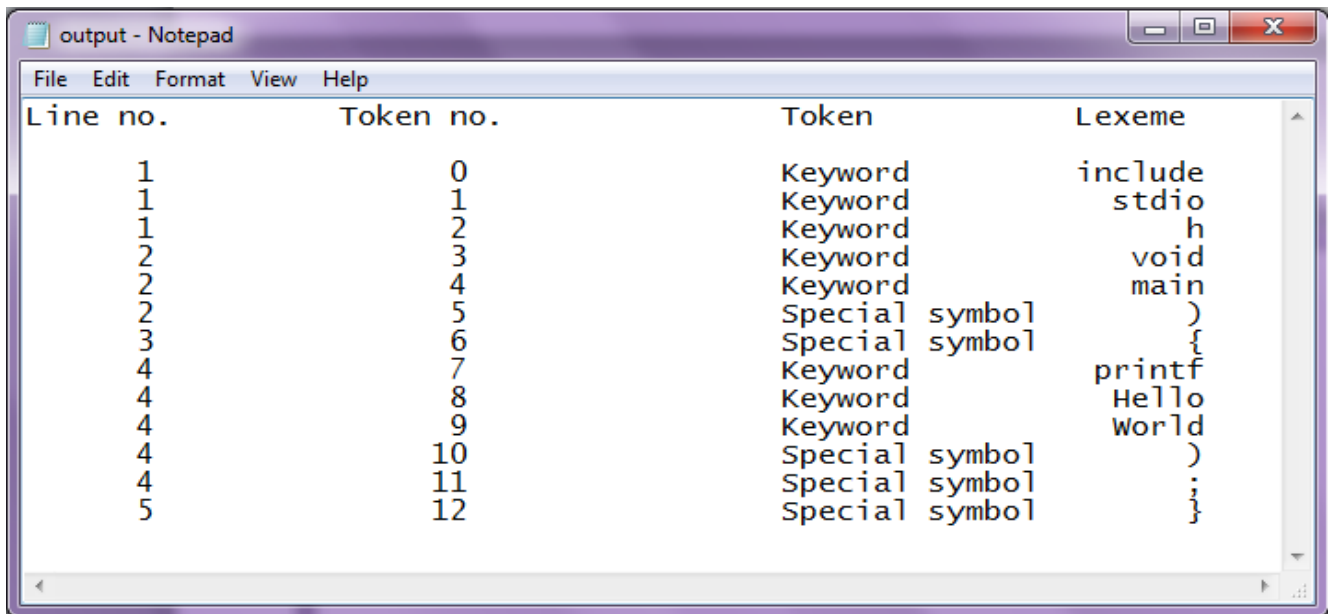
```

```
while(isalnum(ch) && ch!=' ' )
{
    str[i]=ch;
    i++;
    ch=fgetc(input);
}
str[i]='\0';
for(j=0;j<=30;j++)
{
    if(strcmp(str,keyword[j])==0)
    {
        flag=1;
        break;
    }
}
if(flag==1)
{
    fprintf(output,"%7d\t\t %7d\t\t Keyword\t %7s\n",l,t,str);
    t++;
}
else
{
    fprintf(output,"%7d\t\t %7d\t\t Identifier\t %7s\n",l,t,str);
    t++;
}
else if(ch=='\n')
{
    l++;
}
}
fclose(input);
```

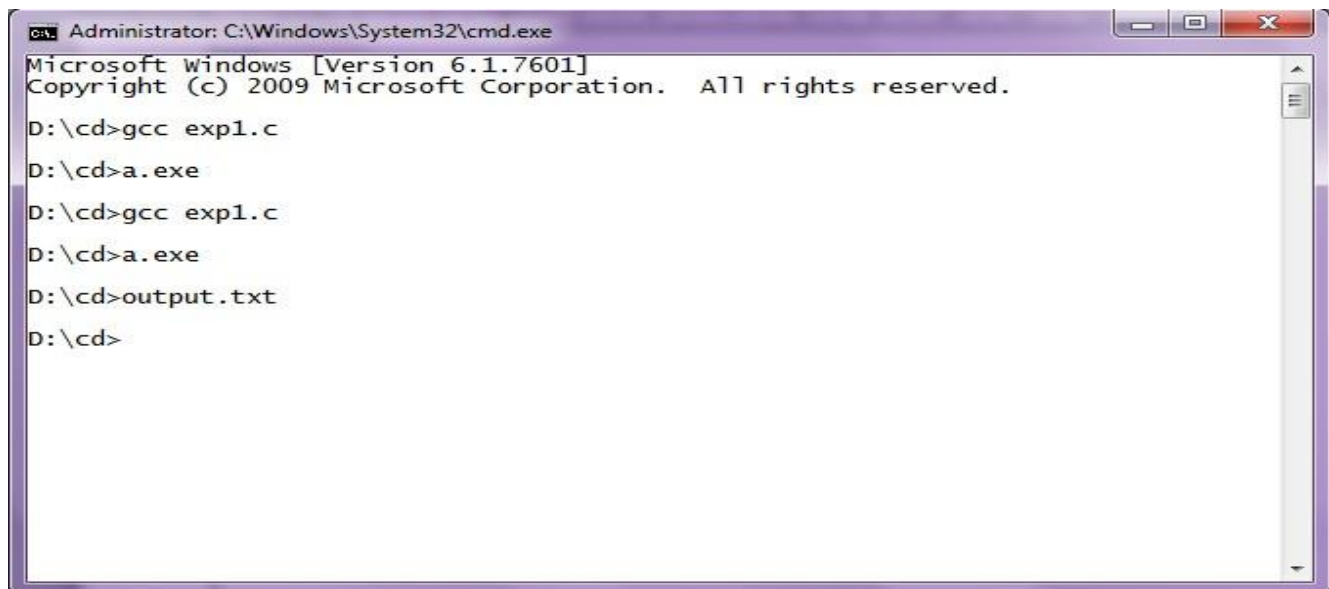
```
fclose(output);  
return 0;  
}
```

input.txt

```
#include<stdio.h>  
void main()  
{  
printf("Hello World");  
}
```



Line no.	Token no.	Token	Lexeme
1	0	Keyword	include
1	1	Keyword	stdio
1	2	Keyword	h
2	3	Keyword	void
2	4	Keyword	main
2	5	Special symbol)
3	6	Special symbol	{
4	7	Keyword	printf
4	8	Keyword	Hello
4	9	Keyword	World
4	10	Special symbol)
4	11	Special symbol	;
5	12	Special symbol	}



```
Administrator: C:\Windows\System32\cmd.exe
Microsoft windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

D:\cd>gcc exp1.c
D:\cd>a.exe
D:\cd>gcc exp1.c
D:\cd>a.exe
D:\cd>output.txt
D:\cd>
```

DEPARTMENT OF CSE		
PERFORMANCE	30	
OBSERVATION	30	
RECORD	40	
TOTAL	100	

Result:

Thus, lexical analyzer to recognize a few patterns in C has been developed and executed successfully.

Ex. No.:1b	IMPLEMENTATION OF SYMBOL TABLE
Date:	

Aim:

To write a program for implementing symbol table using C.

Algorithm:

1. Start the program for performing insert, display, delete, search and modify option in the 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.
7. If it is not found in the symbol table it displays "Label Not found". Else, the symbol is deleted.
8. If the entered choice is 5, the symbol to be modified is searched in the symbol table.

Program:

```
#include<stdio.h>
#include<ctype.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
void main()
{
int i=0,j=0,x=0,n;
void *p,*add[5];
char ch,srch,b[15],d[15],c;
printf("Expression terminated by $:");
while((c=getchar())!='$')
{
```



```
b[i]=c;
i++;
}
n=i-1;
printf("Given Expression:");
i=0;
while(i<=n)
{
printf("%c",b[i]);
i++;
}
printf("\n Symbol Table\n");
printf("Symbol \t addr \t type");
while(j<=n)
{
c=b[j];
if(isalpha(toascii(c)))
{
p=malloc(c);
add[x]=p;
d[x]=c;
printf("\n%c\t%d\tidentifier\n",c,p);
x++;
j++;
}
else
{
ch=c;
if(ch=='+'||ch=='-'||ch=='*'||ch=='=')
{
p=malloc(ch);
add[x]=p;
```

```
d[x]=ch;  
printf("\n %c \t %d \t operator\n",ch,p);  
x++;  
j++;  
}  
}  
}  
}
```

Output:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.2846]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ramya\OneDrive\Desktop>gcc symboltable.c

C:\Users\ramya\OneDrive\Desktop>a.exe
Expression terminated by $:a=b+c*d$
Given Expression:a=b+c*d
Symbol Table
Symbol  addr  type
a       7146776  identifier
=       7146888  operator
b       7146960  identifier
+       7147072  operator
c       7147128  identifier
*       7147240  operator
d       7147296  identifier

C:\Users\ramya\OneDrive\Desktop>
```

DEPARTMENT OF CSE		
PERFORMANCE	30	
OBSERVATION	30	
RECORD	40	
TOTAL	100	

Result:

Thus, a program to implement a symbol table has been written and executed successfully.

Ex. No.:2	LEXICAL ANALYZER USING LEX TOOL
Date:	

Aim:

To write a program for implementing a Lexical analyser using the LEX tool.

Algorithm:

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.

Program:

```
%{  
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("var.c","r");
if(!file)
{
printf("could not open the file");
exit(0);
}
yyin=file;
yylex();
printf("\n");
return(0);
}

int yywrap()
{
return(1);
}

```

Input file: var.c

```

#include<stdio.h>
#include<conio.h>

```

```
void main()
{
int a,b,c;
a=1;
b=2;
c=a+b;
printf("Sum:%d",c);
}
```

Output:

```
C:\Windows\System32\cmd.exe
C:\cd lab>flex lex.l
C:\cd lab>gcc lex.yy.c
C:\cd lab>a.exe

#include<stdio.h> is a preprocessor directive
#include<conio.h> is a preprocessor directive

void is a keyword
FUNCTION
main(
)

BLOCK BEGINS

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

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

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

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

FUNCTION
printf(
"Sum:%d" is a STRING,
c IDENTIFIER
)
;
BLOCK ENDS
```

DEPARTMENT OF CSE		
PERFORMANCE	30	
OBSERVATION	30	
RECORD	40	
TOTAL	100	

Result:

Thus, a program for implementing a Lexical analyzer using LEX tool has been written and executed successfully.

Ex. No.:3a	RECOGNIZE A VALID ARITHMETIC EXPRESSION THAT USES OPERATOR +, -, *, AND / USING YACC
Date:	

Aim:

To write a c program to do exercise on syntax analysis using YACC.

Introduction:

YACC (yet another compiler) is a program designed to produce designed to compile a LALR (1) grammar and to produce the source code of the synthetically analyses of the language produced by the grammar.

Algorithm:

1. Start the program.
2. Write the code for parser. l in the declaration port.
3. Write the code for the 'y' parser.
4. Also write the code for different arithmetical operations.
5. Write additional code to print the result of computation.
6. Execute and verify it.
7. Stop the program.

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
char s[5];
clrscr();
printf("\n Enter any operator:");
gets(s);
switch(s[0])
{
case '>':
if(s[1]=='=')
printf("\n Greater than or equal");
else
printf("\n Greater than");
break;
case '<':
if(s[1]=='=')
printf("\n Less than or equal");
```

```
        else
            printf("\nLess than");
        break;
case '=':
    if(s[1]=='=')
        printf("\nEqual to");
    else
        printf("\nAssignment");
    break;
case '!':
    if(s[1]=='=')
        printf("\nNot Equal");
    else
        printf("\n Bit Not");
    break;
case '&':
    if(s[1]=='&')
        printf("\nLogical AND");
    else
        printf("\n Bitwise AND");
    break;
case '|':
    if(s[1]=='|')
        printf("\nLogical OR");
    else
        printf("\nBitwise OR");
    break;
case '+':
    printf("\n Addition");
    break;
case '-':
    printf("\nSubstraction");
    break;
case '*':
    printf("\nMultiplication");
    break;
case '/':
    printf("\nDivision");
    break;
case '%':
    printf("Modulus");
    break;
default:
    printf("\n Not a operator");
}
getch();
}
```

Output:

```
Enter any operator: *  
Multiplication_
```

DEPARTMENT OF CSE		
PERFORMANCE	30	
OBSERVATION	30	
RECORD	40	
TOTAL	100	

Result:

Thus, the program for the recognize a valid Arithmetic Expression that uses operator +, -, * and / using YACC has been generated successfully.

Ex. No.:3b	RECOGNIZE A VALID VARIABLE WHICH STARTS WITH A LETTER FOLLOWED BY ANY NUMBER OF LETTERS OR DIGITS
Date:	

Aim:

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

Algorithm:

1. Start the program.
2. Declare the required variables and methods.
3. Get the input
4. Then check whether the given string is valid or not
5. Print the result
6. Stop the program

Program:

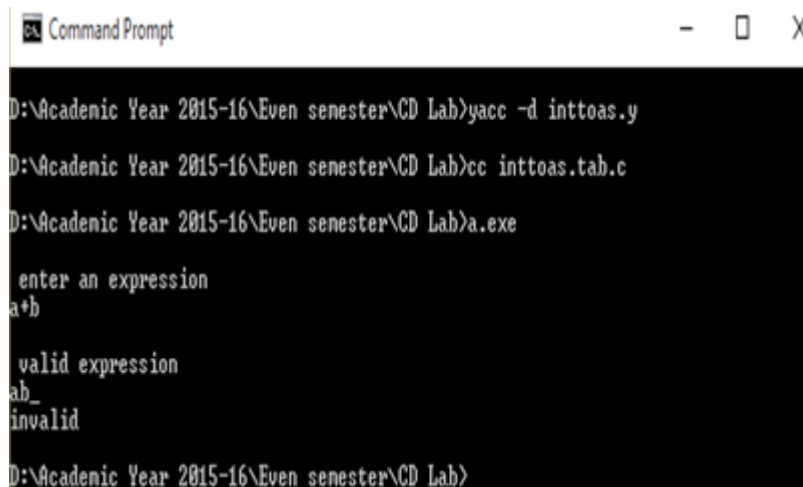
Program Name: 4b.y

```
%{
#include<stdio.h>
#include<ctype.h>
%}
%token LETTER DIGIT
%left '+' '-'
%left '*' '/'
%%
st:st expr '\n'
{printf("\n valid expression \n");}
| st '\n'
|
;
expr:expr '+' expr
|expr '-' expr
|expr '*' expr
|expr '/' expr
|'(' expr ')'
| NUM
| LETTER
;

NUM: DIGIT
|NUM DIGIT
;
%%
```

```
int yylex()
{
    char c;
    while((c=getchar())
    == ' ');
    if(isalpha(c)) return
    LETTER;if(isdigit(c))
    return DIGIT; return(c);
}
int main()
{
    printf("\n enter an
    expression\n");yyparse();
}
int yyerror()
{
    printf("invalid\n");
    return 0;
}
```

Output:



```
D:\Academic Year 2015-16\Even semester\CD Lab>yacc -d inttoas.y
D:\Academic Year 2015-16\Even semester\CD Lab>cc inttoas.tab.c
D:\Academic Year 2015-16\Even semester\CD Lab>a.exe

enter an expression
a+b

valid expression
ab_
invalid

D:\Academic Year 2015-16\Even semester\CD Lab>
```

DEPARTMENT OF CSE		
PERFORMANCE	30	
OBSERVATION	30	
RECORD	40	
TOTAL	100	

Result:

Thus, the program for the recognize a valid variable which starts with a letter followed by any number of letters or digits has been generated successfully.

Ex. No.:3c	RECOGNIZE A VALID CONTROL STRUCTURES SYNTAX OF C LANGUAGE (For loop, while loop, if-else, if-else-if, switch-case, etc.).
Date:	

Aim:

To write a program to recognize control structures syntax of C Language.

Algorithm:

1. Start the program.
2. Implement a lexer to tokenize the input C code.
3. Implement a parser to analyze the tokens produced by the lexer according to the CFG.
4. Start parsing from the start symbol S. Match the tokens according to the CFG rules.
5. Keep track of the current state of the parser, and if at any point you encounter a token that doesn't match the expected next token based on the CFG, raise a syntax error.
6. Continue parsing until you reach the end of the input code.
7. If you successfully parse the entire input without any syntax errors, then defined control structures.
8. Print the result.
9. Stop the program.

Program:

```
#include <stdio.h>
#include <stdbool.h>
#include
<string.h>

bool startsWith(const char *str, const char *prefix)
{
    return strncmp(str, prefix, strlen(prefix)) == 0;
}

int main()
{
    char input[100];
    printf("Enter a C control structure: ");
    fgets(input, sizeof(input), stdin);
    if (startsWith(input, "if (") && strstr(input, "{"))
    {
        printf("Recognized: If-Statement\n");
    }
}
```

```
else if (startsWith(input, "else if (") && strstr(input, ") {"))
{

    printf("Recognized: Else If-Statement\n");
}
else if (startsWith(input, "else {"))
{
    printf("Recognized: Else-Statement\n");
}
else if (startsWith(input, "for (") && strstr(input, ";") && strstr(input, ") {"))
{
    printf("Recognized: For Loop\n");
}
else
{
    printf("Not a recognized C control structure\n");
}
return 0;
}
```


Output:

Enter a C control structure: if (condition) {

Recognized: If-Statement

Enter a C control structure: while (condition) {

Not a recognized C control structure

DEPARTMENT OF CSE		
PERFORMANCE	30	
OBSERVATION	30	
RECORD	40	
TOTAL	100	

Result:

Thus, the program to recognize a valid control structures syntax of C Language has been executed successfully.

Ex. No.:3d	IMPLEMENTATION OF CALCULATOR USING LEX AND YACC
Date:	

Aim:

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

Algorithm:

1. Start the program.
2. Declare the required variables and methods.
3. Get the input using NUM{DIGIT}+(\.{DIGIT})? as two integer value.
4. Then check whether the given number having operator such as + \ - * /
5. After that assign first number to f1 then check if f1==0
6. If it true then op1=atoi(yytext)
7. else if assign the second number to f1 then check f2== -1
8. if it true then op2=atoi(yytext)
9. By using switch check the operator +, -, *, / do the corresponding calc operation.
10. Print the result.
11. Stop the program.

Program:

```
%{
#include<stdio.h> int
op=0,i; float a,b;
}%
dig[0-9]+|([0-9]*)."([0-9]+)
add "+" sub "-" mul "*"
div "/"
pow "^" ln \n
%%
{dig}{digi();}
{add}{op=1;}
{sub}{op=2;}
```

```
{mul}{op=3;}
{div}{op=4;}
{pow}{op=5;}
{ln}{printf("\n the result:%f\n\n",a);}
%%
digi()
{
if(op==0) a=atof(yytext);
else
{
b=atof(yytext);
switch(op)
{
case 1:a=a+b;
break;
case 2:a=a-b;
break;
case 3:a=a*b;
break;
case 4:a=a/b;
break;
case 5:
for(i=a;b>1;b--)
a=a*i;
break;
}
op=0;
}
}
```

```
main(int argv,char *argc[])
```

```
{
```

```
yylex();
```

```
}
```

```
yywrap()
```

```
{
```

```
return 1;
```

```
}
```

Output:

Lex cal.l

Cc lex.yy.c-ll a.out

4*8

The result=32

DEPARTMENT OF CSE		
PERFORMANCE	30	
OBSERVATION	30	
RECORD	40	
TOTAL	100	

Result:

Thus the program to implement calculator using LEX and YACC has been executed successfully.

Ex. No.:4	IMPLEMENTATION OF THREE ADDRESS CODE
Date:	

Aim:

To generate three address code for a simple expression using LEX and YACC.

Algorithm:

1. Start the program.
2. Write the lexical program to get the tokens in the given expression.
3. Write the yacc program to generate the three address code.
 - i. Write the productions for the expression.
 - ii. Call the three address code function. Three address statement is of form $x = y \text{ op } z$, where x , y , and z will have address (memory location). Sometimes a statement might contain less than three references.
 - iii. Call Quadruple function. It consists of 4 fields namely op, arg1, arg2 and result. op denotes the operator and arg1 and arg2 denotes the two operands and result is used to store the result of the expression.
 - iv. Call triple function which consist of only three fields namely op, arg1 and arg2.
4. Print three address code, quadruple and triple statements.
5. Stop the program.

Program:

taclex.l

```
%{
#include "y.tab.h"
%}
%%
[0-9]+? {yylval.sym=(char)yytext[0]; return NUMBER;}
[a-zA-Z]+? {yylval.sym=(char)yytext[0];return LETTER;}
\n {return 0;}
. {return yytext[0];}
%%
yywrap()
{
```

```

    return 1;
}
tacyacc.y
%{
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void ThreeAddressCode();
void triple();
void qudruple();
char AddToTable(char ,char, char);
int ind=0;//count number of lines
char temp = '1';//for t1,t2,t3.....
struct incod
{
char opd1;
char opd2;
char opr;
};
%}
%union
{
char sym;
}
%token <sym> LETTER NUMBER
%type <sym> expr
%left '+'
%left '*' '/'
%left '-'
%%

```

```

statement: LETTER '=' expr ';' {AddToTable((char)$1,(char)$3,'=');}
| expr ';'
;
expr:
    expr '+' expr {$$ = AddToTable((char)$1,(char)$3,'+');}
| expr '-' expr {$$ = AddToTable((char)$1,(char)$3,'-');}
| expr '*' expr {$$ = AddToTable((char)$1,(char)$3,'*');}
| expr '/' expr {$$ = AddToTable((char)$1,(char)$3,'/');}
| '(' expr ')' {$$ = (char)$2;}
| NUMBER {$$ = (char)$1;}
| LETTER {$$ = (char)$1;}
| '-' expr {$$ = AddToTable((char)$2,(char)'\\t','-' );}
;
%%
yyerror(char *s)
{
    printf("%s",s);
    exit(0);
}
struct incod code[20];
char AddToTable(char opd1,char opd2,char opr)
{
    code[ind].opd1=opd1;
    code[ind].opd2=opd2;
    code[ind].opr=opr;
    ind++;
    return temp++;
}
void ThreeAddressCode()
{

```



```

int cnt = 0;
char temp = '1';
printf("\n\n\t THREE ADDRESS CODE\n\n");
while(cnt<ind)
{
if(code[cnt].opr != '=')
printf("t%c : = \t",temp++);
if(isalpha(code[cnt].opd1))
printf(" %c\t",code[cnt].opd1);
else if(code[cnt].opd1 >='1' && code[cnt].opd1 <='9')
printf("t%c\t",code[cnt].opd1);
printf(" %c\t",code[cnt].opr);
if(isalpha(code[cnt].opd2))
printf(" %c\n",code[cnt].opd2);
else if(code[cnt].opd2 >='1' && code[cnt].opd2 <='9')
printf("t%c\n",code[cnt].opd2);
cnt++;
}
}
void quadraple()
{
int cnt = 0;
char temp = '1';
printf("\n\n\t QUADRUPLE CODE\n\n");
while(cnt<ind)
{
printf(" %c\t",code[cnt].opr);
if(code[cnt].opr == '=')
{
if(isalpha(code[cnt].opd2))

```

```

printf(" %c\t \t",code[cnt].opd2);
else if(code[cnt].opd2 >='1' && code[cnt].opd2 <='9')
printf("t%c\t \t",code[cnt].opd2);
printf(" %c\n",code[cnt].opd1);
cnt++;
continue;
}
if(isalpha(code[cnt].opd1))
printf(" %c\t",code[cnt].opd1);
else if(code[cnt].opd1 >='1' && code[cnt].opd1 <='9')
printf("t%c\t",code[cnt].opd1);
if(isalpha(code[cnt].opd2))
printf(" %c\t",code[cnt].opd2);
else if(code[cnt].opd2 >='1' && code[cnt].opd2 <='9')
printf("t%c\t",code[cnt].opd2);
else printf(" %c",code[cnt].opd2);
printf("t%c\n",temp++);
cnt++;
}
}
void triple()
{
int cnt=0;
char temp='1';
printf("\n\n\t TRIPLE CODE\n\n");
while(cnt<ind)
{
printf("(%c) \t",temp);
printf(" %c\t",code[cnt].opr);
if(code[cnt].opr == '=')

```

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

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

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

Output:

```
Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

D:\cd\three_address_code>bison -dy tacyacc.y
D:\cd\three_address_code>flex taclex.l
D:\cd\three_address_code>gcc lex.yy.c y.tab.c
taclex.l:15:1: warning: return type defaults to 'int' [-Wimplicit-int]
15 | yywrap()
   | ~~~~~
D:\cd\three_address_code>a.exe
Enter the Expression : a=b+e;

THREE ADDRESS CODE
t1 := b + e
a  = t1

QUADRUPLE CODE
+   b   e   t1
=   t1      a

TRIPLE CODE
(1) +   b   e
(2) =   (1)

D:\cd\three_address_code>a.exe
```

DEPARTMENT OF CSE		
PERFORMANCE	30	
OBSERVATION	30	
RECORD	40	
TOTAL	100	

Result:

Thus, the intermediate code for a simple expression using LEX and YACC has been generated successfully.

Ex. No.:5	IMPLEMENTATION OF TYPE CHECKING
Date:	

Aim:

To write a C program for implementing type checking for given expression.

Algorithm:

1. Start the program.
2. Include all the header files.
3. Initialize all the functions and variables.
4. Get the expression from the user and separate into the tokens.
5. After separation, specify the identifiers, operators and number.
6. Print the output.
7. Stop the program.

Program:

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

```
for(i=0;i<k;i++)
{
if(b[i]=='/')
{
flag=1;
break;
}
}
for(i=0;i<n;i++)
{
if(b[0]==vari[i])
{
if(flag==1)
{
if(typ[i]=='f')
{
printf("\nthe datatype is correctly defined..!\n");
break;
}
else
{
printf("Identifier %c must be a float type..!\n",vari[i]);
break;
}
}
else
{
printf("\nthe datatype is correctly defined..!\n");
break;
}
}
}
return 0;
}
```

Output:

Enter the number of variables: 2
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 Expression(end with \$): a/b\$

DEPARTMENT OF CSE		
PERFORMANCE	30	
OBSERVATION	30	
RECORD	40	
TOTAL	100	

Result:

Thus the implementation of type checking using C has been executed successfully.

Ex. No.:6	IMPLEMENTATION OF CODE OPTIMIZATION TECHNIQUE
Date:	

Aim:

To implement code optimization techniques using C.

Algorithm:

8. Start the program.
9. Get the expressions from the user as left and right.
10. Eliminate the unused statements.
11. Eliminate common expressions.
12. Print the optimized code.
13. Stop the program.

Program:

code_optimization.c

```
#include<stdio.h>
#include<string.h>
struct op
{
char l;
char r[20];
}
op[10],pr[10];
void main()
{
int a,i,k,j,n,z=0,m,q;
char *p,*l;
char temp,t;
char *tem;
printf("Enter the Number of Values:");
scanf("%d",&n);
```



```
for(i=0;i<n;i++)
{
printf("left: ");
scanf(" %c",&op[i].l);
printf("right: ");
scanf(" %s",&op[i].r);
}
printf("Intermediate Code\n") ;
for(i=0;i<n;i++)
{
printf("%c=",op[i].l);
printf("%s\n",op[i].r);
}
for(i=0;i<n-1;i++)
{
temp=op[i].l;
for(j=0;j<n;j++)
{
p=strchr(op[j].r,temp);
if(p)
{
pr[z].l=op[i].l;
strcpy(pr[z].r,op[i].
r);
z++;
}
}
}
pr[z].l=op[n-1].l;
strcpy(pr[z].r,op[n-1].r);
```

```
z++;
printf("\nAfter Dead Code Elimination\n");
for(k=0;k<z;k++)
{
printf("%c\t=",pr[k].l);
printf("%s\n",pr[k].r);
}
for(m=0;m<z;m++)
{
tem=pr[m].r;
for(j=m+1;j<z;j++)
{
p=strstr(tem,pr[j].r);
if(p)
{
t=pr[j].l;
pr[j].l=pr[m].l;
for(i=0;i<z;i++)
{
l=strchr(pr[i].r,t);
if(l)
{
a=l-pr[i].r;
printf("pos: %d\n",a);
pr[i].r[a]=pr[m].l;
}
}
}
}
}
```

```
printf("Eliminate Common Expression\n");
```

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

```
{
```

```
printf("%c\t=",pr[i].l);
```

```
printf("%s\n",pr[i].r);
```

```
}
```

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

```
{
```

```
for(j=i+1;j<z;j++)
```

```
{
```

```
q=strcmp(pr[i].r,pr[j].r);
```

```
if((pr[i].l==pr[j].l)&&!q)
```

```
{
```

```
pr[i].l='\0';
```

```
}
```

```
}
```

```
}
```

```
printf("Optimized Code\n");
```

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

```
{
```

```
if(pr[i].l!='\0')
```

```
{
```

```
printf("%c=",pr[i].l);
```

```
printf("%s\n",pr[i].r);
```

```
}
```

```
}
```

```
}
```

Output:

```
C:\Windows\System32\cmd.exe

C:\cd lab>gcc code_optimization.c

C:\cd lab>a.exe
Enter the Number of Values:4
left: a
right: 9
left: b
right: c+d
left: e
right: c+d
left: f
right: b+e
Intermediate Code
a=9
b=c+d
e=c+d
f=b+e

After Dead Code Elimination
b      =c+d
e      =c+d
f      =b+e
pos: 2
Eliminate Common Expression
b      =c+d
b      =c+d
f      =b+b
Optimized Code
b=c+d
f=b+b

C:\cd lab>
```

DEPARTMENT OF CSE		
PERFORMANCE	30	
OBSERVATION	30	
RECORD	40	
TOTAL	100	

Result:

Thus the code optimization technique using C has been executed successfully.

Ex. No.:7	IMPLEMENTATION OF BACK END OF THE COMPILER
Date:	

Aim:

To implement the back end of the compiler which takes the three address code as input and produces the 8086 assembly language instructions as output.

Algorithm:

1. Start the program
For each three address statement of the form $x := y \text{ op } z$, perform the following actions:
2. Determine the location L where the result of the computation $y \text{ op } z$ should be stored. L will usually be a register, but it could also be a memory location.
3. Consult the address descriptor for y to determine y' , the current location of y. Prefer the register for y' if the value of y is currently both in memory and a register. If the value of y is not already in L, generate the instruction `MOV y' , L` to place a copy of y in L.
4. Generate the instruction `OP z' , L` where z' is a current location of z. Again, prefer a register to a memory location if z is in both. Update the address descriptor of x to indicate that x is in location L. If L is a register, update its descriptor to indicate that it contains the value of x, and remove x from all other register descriptors.
5. If the current values of y and/or z have no next users, are not live on exit from the block, and are in register descriptor to indicate that, after execution of $x := y \text{ op } z$, those registers no longer will contain y and/or z, respectively.
6. Print the output.
7. Stop the program.

Program:

Back_end.c

```
#include<stdio.h>
#include<stdio.h>
#include<string.h>
void main()
{
```

```
char icode[10][30],str[20],opr[10];
int i=0;
printf("\n Enter 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:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.2846]
(c) Microsoft Corporation. All rights reserved.

C:\cd lab>gcc back_end.c

C:\cd lab>a.exe

Enter the set of intermediate code (terminated by exit):
a=a*b
c=f*h
g=a+c
exit

target code generation
*****
Mov a,R0
MULb,R0
Mov R0,a
Mov f,R1
MULh,R1
Mov R1,c
Mov a,R2
ADDc,R2
Mov R2,g
C:\cd lab>
```

DEPARTMENT OF CSE		
PERFORMANCE	30	
OBSERVATION	30	
RECORD	40	
TOTAL	100	

Result:

Thus, the back end of the compiler which takes the three address code as input and produces the 8086 assembly language instructions as output has been implemented successfully.