MODEL ENGINEERING COLLEGE, THRIKKAKARA

DEPARTMENT OF COMPUTER ENGINEERING
B.TECH IN COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL

CSL411 COMPILER LAB

Govt.Model Engineering College,Thrikkakara

Managed by Institute of Human Resources Development
A Govt.of Kerala Establishment
Department of Computer Engineering
B.Tech in Computer Science and Engineering

INDEX

Govt.Model Engineering College,Thrikkakara

Managed by Institute of Human Resources Development
A Govt.of Kerala Establishment
Department of Computer Engineering
B.Tech in Computer Science and Engineering

	Vision	**Mission**
INSTITUTION (Model Engg. College)	Evolve into an academy of excellence to serve the knowledge society.	M1: Implement quality education through Teaching Learning Process. M2: Inculcate culture of technical innovations and creativity. M3: Instill high standards of professional ethics and social values.
DEPARTMENT (Computer Engineering)	Evolve into a center of excellence to serve the emerging knowledge society.	M1: Impart quality education to the graduate and undergraduate students in Computer Science and Engineering. M2: Inculcate students with technical knowledge and human values to create socially committed Engineers. M3: Empower the students to succeed in innovative research and developments to serve the computational needs of the society.

	Govt.Model Engineering College,Thrikkakara **Managed by Institute of Human Resources Development** **A Govt.of Kerala Establishment** **Department of Computer Engineering** **B.Tech in Computer Science and Engineering**

PROGRAM OUTCOMES

PO1:Engineering knowledge	**Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.**
PO 2: Problem analysis	**Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.**
PO3:Design/development of solutions	**Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.**
PO4:Conduct investigations of complex problems	**Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.**
PO 5: Modern tool usage	**Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.**

PO 6: The engineer and society	**Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.**
PO 7: Environment and sustainability	**Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.**
PO 8: Ethics	**Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.**
PO 9: Individual and team work	**Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.**
PO 10: Communication	**Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.**

PO11:Project management and finance	**Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.**
PO12: Life-long learning	**Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.**

PROGRAM SPECIFIC OUTCOMES

PSO1: Hardware & Software	**An ability to analyze, design, and develop system software, secure application software, intelligent systems, computer architecture, and network-based computing solutions.**
PSO2:Problem Solving	**An ability to analyze & design algorithms, and implement the solutions incorporating various programming concepts.**
PSO3:Project Management	**An ability to apply diverse software project development approaches to tackle real time problems.**

PROGRAM EDUCATIONAL OBJECTIVES

PEO1:Lifelong Learning	**To produce graduates with solid foundation in Computer Science & Engineering and broad knowledge in mathematics, applied science and basic engineering with competence for higher studies and to pursue a profession in computing.**
PEO2:Entrepreneurship	**To impart an educational foundation that enables them to be good entrepreneurs and be adaptive to the advancements in the latest technologies.**
PEO3:Social Commitment & Leadership	**To make them capable to function in multi-disciplinary teams, ethically and responsibly, contributing to the information technology requirements of the society.**

LAB RULES

1. **BE PUNCTUAL FOR THE LAB.**
2. **STUDENTS SHOULD LEAVE THEIR FOOTWEAR OUTSIDE THE LAB IN THE SPACE PROVIDED.**
3. **KEEP YOUR BELONGINGS OUTSIDE THE LAB.**
4. **RECORD ENTRY AND EXIT TIME IN THE LOG-REGISTER WHILE USING THE LAB.**
5. **COME WELL PREPARED WITH RECORD FOR DOING THE EXPERIMENT.**
6. **AVOID STEPPING ON ELECTRICAL WIRES OR COMPUTER CABLES.**
7. **DO NOT SHIFT OR INTERCHANGE ANY PART OF THE COMPUTER WITH ANOTHER.**
8. **FOR ANY HARDWARE PROBLEM, REPORT TO THE LAB-IN-CHARGE / LAB STAFF.**

9. **STUDENTS SHOULD MAINTAIN SILENCE AND DECORUM IN THE LAB.**
10. **STUDENTS SHOULD LOG OFF THE COMPUTERS AND ARRANGE THEIR SEATS BEFORE LEAVING THE LABORATORY.**
11. **VIOLATION OF LAB RULES WILL NECESSITATES IN THE LOSS OF YOUR LAB PRIVILEGES**

INSTRUCTIONS FOR MAINTAINING THE LAB RECORD

1. The index page should be filled properly by writing the corresponding experiment number, name and date on which the experiment was performed.
2. Every experiment conducted in the lab should be noted in the fair record.
3. For every experiment in the fair record the right-hand page should contain

    - **Title:** The experiment heading on top of the page in capital letters with experiment number and date of experiment.
    - **Aim:** The aim of experiment in one or two sentences clearly.
    - **Algorithm:** Steps for doing the experiment.
    - **Result:** The result of the experiment must be summarized.

4. The left-hand page should contain

    - A **print out of the code** used for the experiment
    - **Sample output** obtained for a set of input.

	Govt.Model Engineering College,Thrikkakara **Managed by Institute of Human Resources Development** **A Govt.of Kerala Establishment** **Department of Computer Engineering** **B.Tech in Computer Science and Engineering**

Course code	Course name	L-T-P	Year of Introduction
CSL411	COMPILER LAB	0-0-3-2	2019

Pre-requisite : CSL331 System Software Lab,CST 301 Formal languages and Automata Theory, CST 302 Compiler Design

SYLLABUS

1. Implementation of lexical analyzer using the tool LEX.
2. Implementation of Syntax analyzer using the tool YACC.
3. Application problems using NFA and DFA.
4. Implement Top-Down Parser.
5. Implement Bottom-up parser.
6. Simulation of code optimization Techniques.
7. Implement Intermediate code generation for simple expressions.
8. Implement the back end of the compiler.

Course Objectives:

To implement the different Phases of the compiler.
To implement and test simple optimization techniques.
To give exposure to compiler writing tools.

Expected Outcome:

The Student will be able to :
i. Implement the techniques of Lexical Analysis and Syntax Analysis.
ii. Apply the knowledge of Lex & Yacc tools to develop programs.
iii. Generate intermediate code.
iv. Implement Optimization techniques and generate machine level code.

PRACTICE QUESTIONS

List of Exercises/Experiments :

1. Design and implement a lexical analyzer using C language to recognize all valid tokens in the input program. The lexical analyzer should ignore redundant spaces, tabs and newlines. It should also ignore comments.

2. Implement a Lexical Analyzer for a given program using Lex Tool.

3. Write a lex program to display the number of lines, words and characters in an input text.

4. Write a LEX Program to convert the substring abc to ABC from the given input string.

5. Write a lex program to find out the total number of vowels and consonants from the given input string.

6. Generate a YACC specification to recognize a valid arithmetic expression that uses operators +, – , *,/ and parenthesis. COMPUTER SCIENCE AND ENGINEERING

7. Generate a YACC specification to recognize a valid identifier which starts with a letter followed by any number of letters or digits.

8. Implementation of Calculator using LEX and YACC

9. Convert the BNF rules into YACC form and write code to generate abstract syntax tree.

10. Write a program to find ε – closure of all states of any given NFA with ε transition.
11. Write a program to convert NFA with ε transition to NFA without ε transition.

12. Write a program to convert NFA to DFA.

13. Write a program to minimize any given DFA.

14. Write a program to find First and Follow of any given grammar.

15. Design and implement a recursive descent parser for a given grammar.

16. Construct a Shift Reduce Parser for a given language.

17. Write a program to perform constant propagation.

18. Implement Intermediate code generation for simple expressions.

19. 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 an 8086 assembler. The target assembly instructions can be simple move, add, sub, jump etc.

CSL411 COMPILER LAB COURSE OUTCOMES

CSL411.1	Implement lexical analyzer using the tool LEX. (Cognitive Knowledge Level: Apply)
CSL411.2	Implement Syntax analyzer using the tool YACC. (Cognitive Knowledge Level: Apply)
CSL411.3	Design NFA and DFA for a problem and write programs to perform operations on it. (Cognitive Knowledge Level: Apply)
CSL411.4	Design and Implement Top-Down parsers. (Cognitive Knowledge Level: Apply)
CSL411.5	Design and Implement Bottom-Up parsers. (Cognitive Knowledge Level: Apply)
CSL411.6	Develop intermediate code for simple expressions. (Cognitive Knowledge Level: Apply)

CO-PO MAPPING

Course outcomes	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 1O	PO 11	PO 12
CSL411.1	3	3	2	2	0			0		0		2
CSL411.2	3	3	2	3	0			0		0		2
CSL411.3	3	3	3	3				0		0		2
CSL411.4	3	3	3	3				0		0		2
CSL411.5	3	3	2	2				0		0		2
CSL411.6	3	3	3	3				0		0		2

CO-PSO MAPPING

Course outcomes	PSO 1	PSO 2	PSO 3
CSL411.1	3	3	
CSL411.2	3	3	
CSL411.3	3	3	
CSL411.4	3	3	
CSL411.5	3	3	
CSL411.6	3	3	

List of Experiments-CO Mapping

1. Design and implement a lexical analyzer using C language to recognize all valid tokens in the input program. The lexical analyzer should ignore redundant spaces, tabs and newlines. It should also ignore comments.

2. Implement a Lexical Analyzer for a given program using Lex Tool.

3. Write a lex program to display the number of lines, words and characters in an input text.

4. Write a LEX Program to convert the substring abc to ABC from the given input string.

5. Write a lex program to find out the total number of vowels and consonants from the given input string.

6. Generate a YACC specification to recognize a valid arithmetic expression that uses operators +, – , *,/ and parenthesis. COMPUTER SCIENCE AND ENGINEERING

7. Generate a YACC specification to recognize a valid identifier which starts with a letter followed by any number of letters or digits.

8. Implementation of Calculator using LEX and YACC

9. Convert the BNF rules into YACC form and write code to generate abstract syntax tree.

10. Write a program to find ε – closure of all states of any given NFA with ε transition.

11. Write a program to convert NFA with ε transition to NFA without ε transition.

12. Write a program to convert NFA to DFA.

13. Write a program to minimize any given DFA.

14. Write a program to find First and Follow of any given grammar.

15. Design and implement a recursive descent parser for a given grammar.

16. Construct a Shift Reduce Parser for a given language.

17. Write a program to perform constant propagation.

18. Implement Intermediate code generation for simple expressions.

19. 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 an 8086 assembler. The target assembly instructions can be simple move, add, sub, jump etc.

Reference Books:

1. Aho A Ravi Sethi and J D Ullman, Compilers Principles Techniques and Tools, Addison Wesley
2. Kenneth C Louden, “Compiler Construction Principles and Practice”, Cenage Learning Indian Edition
3. D M Dhamdhare, System programming and operating system, Tata McGraw Hill & Company
4. Tremblay and Sorenson, The Theory and Practice of Compiler Writing - Tata

McGraw Hill & Company

Evaluation Methods For Laboratory Experiments:

Assessment Methods	Continuous Evaluation				Internal
Rubrics	R1	R2	R3	R4	R5
Marks	15	10	5	15	15
Total	60				

Criteria	Parameters evaluated	Performance indicators		
		Beginning(1)	Satisfactory(2)	Exemplary(3)

Algorithm And Program(R1)	• Clarity of the problem • Neatness and completeness in the algorithm and program	Lack of clarity Of the problem. Algorithm and program are not accurate.	Demonstrates fair understanding of the concepts and problem. Algorithm and program needs minor correction.	Demonstrates appropriate understanding of the concepts and problem. The algorithm and the program are accurate and neat.
Viva(R2)	• Knowledge of concepts and procedure. • Pre-requisite knowledge needed for implementing the problem.	Unable to articulate the concepts and does not answer most of the questions.	Articulates fairly well and answers most of the questions correctly.	Good command over the concepts associated with the problem and answers all questions.

Record(R3)	• Promptness in submission. • Neat documentation. • Accurate output	Incomplete /Late submission of the record.	Late submission. Record documentation is fair.	Record submission on time. Record documentation is complete and neat with perfect outputs.

Time utilization and completion(R4)	• Follow the instructions given to complete the lab • Completion of the lab in allotted time.	Unable to follow the verbal and written instructions to complete the lab. Failed to complete even a part of the lab in allotted time.	Unable to follow some of the verbal and written instructions to complete the requirements of the lab The student failed to complete the entire lab in the allotted amount of time	Able to follow the verbal and written instructions to successfully complete requirements of the lab. The student completed the lab in its entirety in the allotted amount of time
Continuous assessment Test (R5)	The assessment test shall be conducted for 100 marks, which will be converted to out of 15. The marks will be distributed as, Algorithm - 30 marks, Program - 20 marks, Output – 20 marks and Viva - 30 marks.			

SAMPLE PROGRAMS

1. Design and implement a lexical analyzer for given language using C and the lexical analyzer should ignore redundant spaces, tabs and newlines.

Program

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

int isKeyword(char buffer[]){
 char keywords[32][10] =
{"auto","break","case","char","const","continue","default",

"do","double","else","enum","extern","float","for","goto",

"if","int","long","register","return","short","signed",

"sizeof","static","struct","switch","typedef","union",
                                        "unsigned","void","volatile","while"};
 int i;
 for(i = 0; i < 32; ++i){
        if(strcmp(keywords[i], buffer) == 0){
                return 1;
        }
 }
```

```
 return 0;
}

int main() {
 char c, buffer[31], operators[] = "+-*/%=";
 FILE *fp;
 int i, j=0;

 fp = fopen("Program","r");

 if(fp == NULL){
        printf("Error while opening the file\n");
        exit(0);
 }

 while((c = fgetc(fp)) != EOF) {
                for(i = 0; i < 6; ++i){
                        if(c == operators[i])
                                printf("%c is operator\n", c);
                }

                if(isalnum(c)) {
                        buffer[j++] = c;
                } else if((c == ' ' || c =='\t' || c == '\n') && (j != 0)) {
         buffer[j] = '\0';
         j = 0;
```

```
                    if(isKeyword(buffer) == 1)
                        printf("%s is keyword\n", buffer);
                    else
                        printf("%s is identifier\n", buffer);
                }
        }
        fclose(fp);
        return 0;
        }
```

Output

Input-void main()

```
{
int a, b, c;
c = a + b;
        }
```

```
christy@christy-VirtualBox:~$ gcc lex.c
christy@christy-VirtualBox:~$ ./a.out
void is keyword
main is identifier
int is keyword
a is identifier
b is identifier
c is identifier
c is identifier
= is operator
a is identifier
+ is operator
b is identifier
christy@christy-VirtualBox:~$
```

2. Implement a Lexical Analyzer for a given program using Lex Tool

Program:

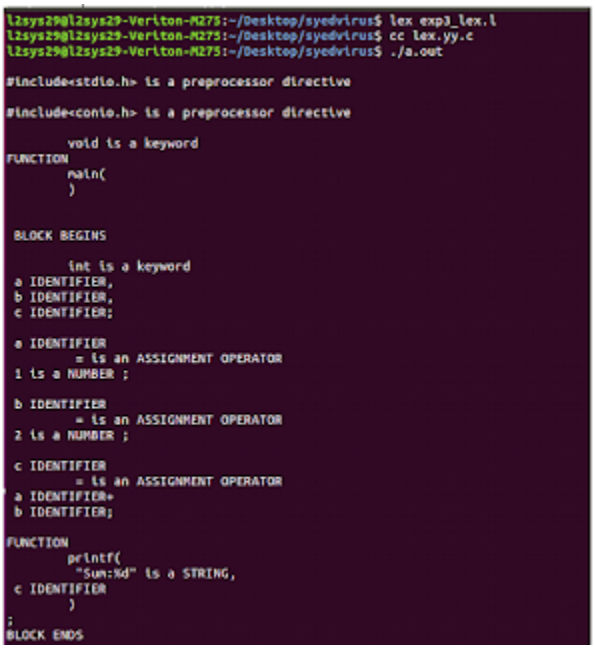
```
%{
int COMMENT=0;
%}
identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* {printf("\n%s is a preprocessor directive",yytext);}
int |
float |
char |
double |
while |
for |
struct |
typedef |
```

```
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);
}
```

OUTPUT:


l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ lex exp3_lex.l
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ cc lex.yy.c
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ ./a.out
#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

3.Write a lex program to display the number of lines, words and characters in an input text.

Program:

```
/*lex code to count the number of lines,
 tabs and spaces used in the input*/

%{
#include<stdio.h>
int lc=0, sc=0, tc=0, ch=0; /*Global variables*/
%}

/*Rule Section*/
%%
\n lc++; //line counter
([ ])+ sc++; //space counter
\t tc++; //tab counter
. ch++;  //characters counter
%%

int main()
{
 // The function that starts the analysis
 yylex();

 printf("\nNo. of lines=%d", lc);
 printf("\nNo. of spaces=%d", sc);
```

```
printf("\nNo. of tabs=%d", tc);
printf("\nNo. of other characters=%d", ch);

}
```

OUTPUT:

```
lab2@csit2pc23:~$ lex gfg.l
lab2@csit2pc23:~$ cc lex.yy.c -lfl
lab2@csit2pc23:~$ ./a.out
Geeks for       Geeks
gfg   gfg

No. of lines=2
No. of spaces=4
No. of tabs=1
No. of other characters=19lab2@csit2pc23:~$
```

4.Write a LEX Program to convert the substring abc to ABC from the given input string.

Program

```
/* lex code to check for characters other that
alphabets in a given string */

%{
int flag = 0;
%}

%%
[\n] {
 flag==0?printf("Only alphabets present\n"):
 printf("Other characters are also present\n");
 flag = 0;
}

[^a-zA-Z] {flag = 1;}

. {}
%%

int yywrap(void) {}

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

```
return 0;
}
```

OUTPUT:

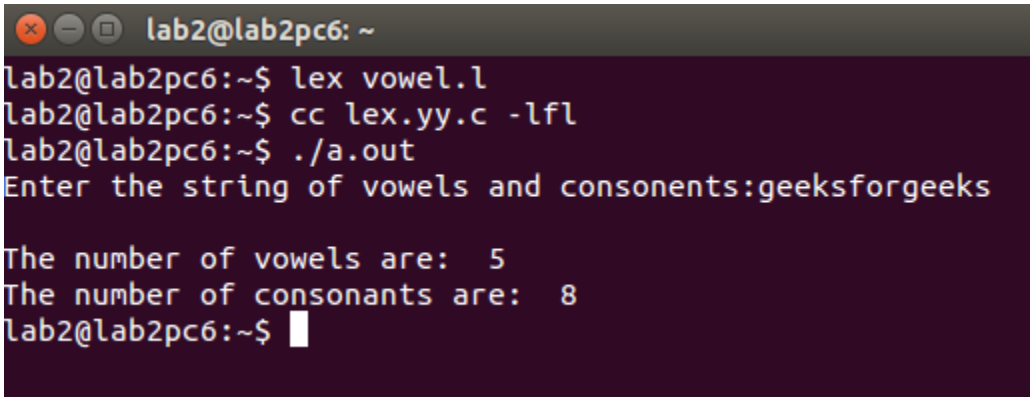
```
abhijeet@fornax:~/gfg$ lex q3.l
abhijeet@fornax:~/gfg$ cc lex.yy.c
abhijeet@fornax:~/gfg$ ./a.out
geeksforgeeks
only alphabets present
geek12
other characters are also present
1youareawesome
other characters are also present
youareawesome
only alphabets present
```

5.Write a lex program to find out total number of vowels and consonants from the given input sting.

Program:

```
%{
 int vow_count=0;
 int const_count =0;
%}

%%
[aeiouAEIOU] {vow_count++;}
[a-zA-Z] {const_count++;}
%%
int yywrap(){}
int main()
{
 printf("Enter the string of vowels and consonents:");
 yylex();
 printf("Number of vowels are: %d\n", vow_count);
 printf("Number of consonants are: %d\n", const_count);
 return 0;
}
```

OUTPUT:


lab2@lab2pc6: ~
lab2@lab2pc6:~$ lex vowel.l
lab2@lab2pc6:~$ cc lex.yy.c -lfl
lab2@lab2pc6:~$ ./a.out
Enter the string of vowels and consonents:geeksforgeeks
The number of vowels are: 5
The number of consonants are: 8
lab2@lab2pc6:~$

6. Generate a YACC specification to recognize a valid arithmetic expression that uses
operators +, – , *,/ and parenthesis.

Program:

Lexical Analyzer Source Code:

```
%{
 /* Definition section*/
 #include "y.tab.h"
 extern yylval;
}%

%%
[0-9]+ {
   yylval = atoi(yytext);
   return NUMBER;
   }

[a-zA-Z]+ { return ID; }
[ \t]+   ; /*For skipping whitespaces*/

\n    { return 0; }
.    { return yytext[0]; }

%%
```

PARSER SOURCE CODE:

```
%{
 /* Definition section */
#include <stdio.h>
%}

%token NUMBER ID
// setting the precedence
// and associativity of operators
%left '+' '-'
%left '*' '/'

/* Rule Section */
%%
E : T  {
    printf("Result = %d\n", $$);
    return 0;
   }

T :
 T '+' T { $$ = $1 + $3; }
 | T '-' T { $$ = $1 - $3; }
 | T '*' T { $$ = $1 * $3; }
 | T '/' T { $$ = $1 / $3; }
 | '-' NUMBER { $$ = -$2; }
 | '-' ID { $$ = -$2; }
 | '(' T ')' { $$ = $2; }
```

```
 | NUMBER { $$ = $1; }
 | ID { $$ = $1; };
% %

int main() {
 printf("Enter the expression\n");
 yyparse();
}

/* For printing error messages */
int yyerror(char* s) {
 printf("\nExpression is invalid\n");
}
```

OUTPUT:

```
aashutosh@ubuntu:~/Desktop/lex-Yacc_prgms$ lex evaluate_exp.l
aashutosh@ubuntu:~/Desktop/lex-Yacc_prgms$ yacc -d evaluate_exp.y
aashutosh@ubuntu:~/Desktop/lex-Yacc_prgms$ cc lex.yy.c y.tab.c -ll
evaluate_exp.l:4:8: warning: type defaults to ‘int’ in declaration of ‘yylval’
[-Wimplicit-int]
 extern yylval;
        ^~~~~~
y.tab.c: In function ‘yyparse’:
y.tab.c:1124:16: warning: implicit declaration of function ‘yylex’ [-Wimplicit-
function-declaration]
       yychar = yylex ();
                ^~~~~
y.tab.c:1314:7: warning: implicit declaration of function ‘yyerror’; did you me
an ‘yyerrok’? [-Wimplicit-function-declaration]
       yyerror (YY_("syntax error"));
       ^~~~~~~
       yyerrok
aashutosh@ubuntu:~/Desktop/lex-Yacc_prgms$ ./a.out
Enter the expression
7*(5-3)/2
Result = 7
aashutosh@ubuntu:~/Desktop/lex-Yacc_prgms$ ./a.out
Enter the expression
6/((3-2)*(-5+2))
Result = -2
aashutosh@ubuntu:~/Desktop/lex-Yacc_prgms$
```

7. Generate a YACC specification to recognize a valid identifier which starts with a letter followed by any number of letters or digits.

Program:

LEX PART:

```
%{
        #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;
```

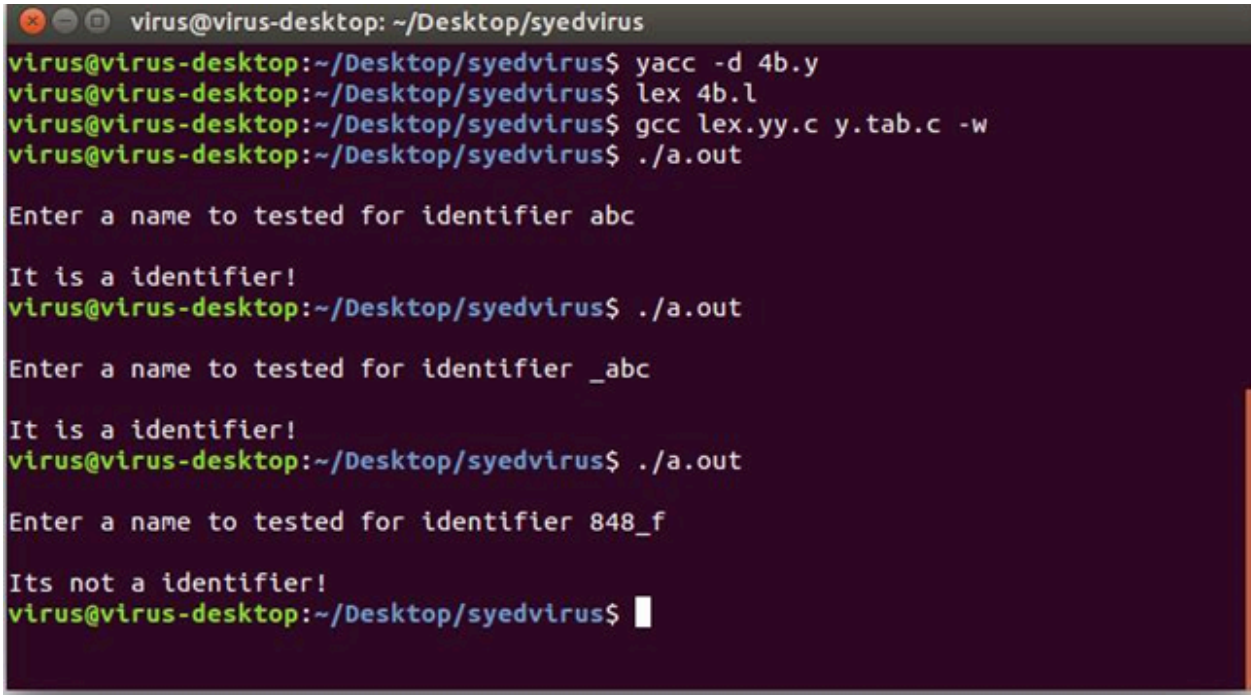
```
}
```

YACC PART:

```
%{
   #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:


virus@virus-desktop: ~/Desktop/syedvirus
virus@virus-desktop:~/Desktop/syedvirus$ yacc -d 4b.y
virus@virus-desktop:~/Desktop/syedvirus$ lex 4b.l
virus@virus-desktop:~/Desktop/syedvirus$ gcc lex.yy.c y.tab.c -w
virus@virus-desktop:~/Desktop/syedvirus$ ./a.out
Enter a name to tested for identifier abc
It is a identifier!
virus@virus-desktop:~/Desktop/syedvirus$ ./a.out
Enter a name to tested for identifier _abc
It is a identifier!
virus@virus-desktop:~/Desktop/syedvirus$ ./a.out
Enter a name to tested for identifier 848_f
Its not a identifier!
virus@virus-desktop:~/Desktop/syedvirus$

8. Implementation of Calculator using LEX and YACC

Program

Code.l

```
%{
   #include<stdio.h>
   #include "y.tab.h"
   extern int yylval;
%}

%%
[0-9]+  {
          yylval = atoi(yytext);
          return digit;
        }
[\t ]+  ;
.       return yytext[0];
\n      return 0;
%%

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

code.y

```
%{
   #include<stdio.h>
%}
%token digit
```

```
%%
start:  E           { printf("%d\n", $1); }
        ;
E:      E '+' T          { $$ = $1 + $3; }
        | E '-' T        { $$ = $1 - $3; }
        | T
        ;
T:      T '*' F          { $$ = $1 * $3; }
        | T '/' F        {  if($3) $$ = $1 / $3;
                            else return yyerror("Divide by zero"); }
        | F
        ;
F:      '(' E ')'        { $$ = $2; }
        | digit          { $$ = $1; }
        ;
%%

int yyerror(char* s) {
    fprintf(stderr, "%s\n", s);
    return 0;
}

int main() {
    printf("Input the expression: ");
    yyparse();
}
```

Output

```
christy@christy-VirtualBox:~$ lex calculator.l
christy@christy-VirtualBox:~$ bison -dy calculator.y
christy@christy-VirtualBox:~$ gcc lex.yy.c y.tab.c
y.tab.c: In function ‘yyparse’:
y.tab.c:1017:16: warning: implicit declaration of function ‘yylex’ [-Wimplicit-
function-declaration]
 1017 |       yychar = yylex ();
      |                ^~~~~
calculator.y:15:45: warning: implicit declaration of function ‘yyerror’; did yo
u mean ‘yyerrok’? [-Wimplicit-function-declaration]
   15 |                                 else return yyerror("Divide by zero");
}
      |                                             ^~~~~~~
      |                                             yyerrok
christy@christy-VirtualBox:~$ ./a.out
Input the expression: 6-3
3
christy@christy-VirtualBox:~$
```

9. Convert the BNF rules into YACC form and write code to generate abstract syntax tree

Program:

Code.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 yywrap(void){};
```

code.y

```
%{
#include<string.h>
#include<stdio.h>
#include<stdlib.h>

int yylex();
int yyerror();
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;

void push(int data)
{
    stk.top++;
    if(stk.top==100)
    {
        printf("\n Stack overflow\n");
        exit(0);
    }
    stk.items[stk.top]=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);
}

int pop()
{
    int data;
    if(stk.top==-1)
    {
        printf("\n Stack underflow\n");
        exit(0);
    }
    data=stk.items[stk.top--];
    return data;
}


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

Output

```
christy@christy-VirtualBox:~$ ./a.out input.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
                ----------------------------

christy@christy-VirtualBox:~$
```

Rhythmbox

10. Write program to find ε – closure of all states of any given NFA with ε – transition

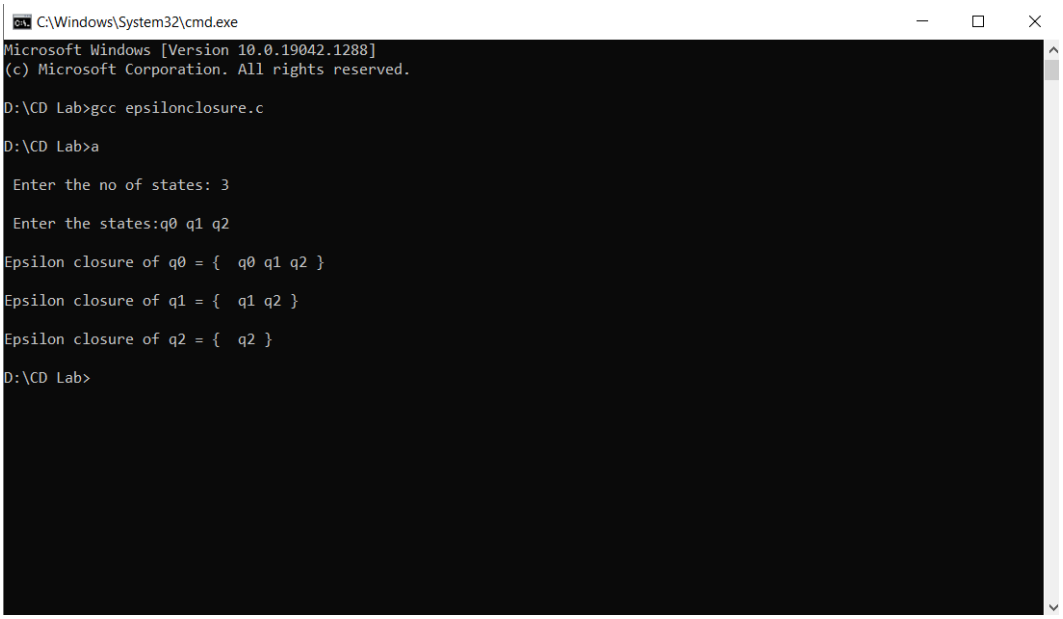
Program

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

char result[20][20],copy[3],states[20][20];

void add_state(char a[3],int i){
 strcpy(result[i],a);
}

void display(int n){
 int k=0;
 printf("\nEpsilon closure of %s = { ",copy);
 while(k < n){
        printf(" %s",result[k]);
        k++;
 }
 printf(" } \n");
}

int main(){
   FILE *INPUT;
   INPUT=fopen("input.txt","r");
   char state[3];
```

```
  int end,i=0,n,k=0;
  char state1[3],input[3],state2[3];
  printf("\n Enter the no of states: ");
  scanf("%d",&n);
  printf("\n Enter the states:");
  for(k=0;k<3;k++){
        scanf("%s",states[k]);

}

for( k=0;k<n;k++){
        i=0;
        strcpy(state,states[k]);
        strcpy(copy,state);
        add_state(state,i++);
        while(1){
                end = fscanf(INPUT,"%s%s%s",state1,input,state2);
                if (end == EOF ){
                        break;
                }

                if( strcmp(state,state1) == 0 ){
                        if( strcmp(input,"e") == 0 ) {
                                add_state(state2,i++);
                                strcpy(state, state2);
                        }
                }
```

```
        }
        display(i);
        rewind(INPUT);
 }

   return 0;
}
```

Output



11. Write program to convert NFA with ε transition to NFA without ε transition.

Program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
	int st;
	struct node *link;
};

void findclosure(int,int);
void insert_trantbl(int ,char, int);
int findalpha(char);
void findfinalstate(void);
void unionclosure(int);
void print_e_closure(int);
static int set[20],nostate,noalpha,s,notransition,nofinal,start,finalstate[20],c,r,buffer[20];
char alphabet[20];
static int e_closure[20][20]={0};
struct node * transition[20][20]={NULL};
void main()
{
	int i,j,k,m,t,n;

	struct node *temp;
```

```
        printf("enter the number of alphabets?\n");
        scanf("%d",&noalpha);
        getchar();
        printf("NOTE:- [ use letter e as epsilon]\n");

       printf("NOTE:- [e must be last character ,if it is present]\n");

       printf("\nEnter alphabets?\n");
       for(i=0;i<noalpha;i++)
       {

              alphabet[i]=getchar();
              getchar();
      }
      printf("Enter the number of states?\n");
      scanf("%d",&nostate);
      printf("Enter the start state?\n");
      scanf("%d",&start);
      printf("Enter the number of final states?\n");
      scanf("%d",&nofinal);
      printf("Enter the final states?\n");
      for(i=0;i<nofinal;i++)
            scanf("%d",&finalstate[i]);
       printf("Enter no of transition?\n");
      scanf("%d",&notransition);
                  printf("NOTE:- [Transition is in the form--> qno     alphabet
qno]\n",notransition);
```

```
printf("NOTE:- [States number must be greater than zero]\n");
printf("\nEnter transition?\n");
for(i=0;i<notransition;i++)
{


        scanf("%d %c%d",&r,&c,&s);
        insert_trantbl(r,c,s);

}

printf("\n");

for(i=1;i<=nostate;i++)
{
        c=0;
        for(j=0;j<20;j++)

        {
                buffer[j]=0;
                 e_closure[i][j]=0;
        }
        findclosure(i,i);
}
printf("Equivalent NFA without epsilon\n");
printf("-------------------------------\n");
printf("start state:");
```

```
print_e_closure(start);
printf("\nAlphabets:");
for(i=0;i<noalpha;i++)
        printf("%c ",alphabet[i]);
printf("\n States :" );
for(i=1;i<=nostate;i++)
        print_e_closure(i);

printf("\nTnransitions are...:\n");

for(i=1;i<=nostate;i++)
{

        for(j=0;j<noalpha-1;j++)
        {
                for(m=1;m<=nostate;m++)
                        set[m]=0;
                for(k=0;e_closure[i][k]!=0;k++)
                {

                        t=e_closure[i][k];
                        temp=transition[t][j];
                        while(temp!=NULL)
                        {

                                unionclosure(temp->st);
                                temp=temp->link;
```

```
                        }
                }
                printf("\n");
                print_e_closure(i);
                printf("%c\t",alphabet[j]   );
                printf("{");
                for(n=1;n<=nostate;n++)
                {
                        if(set[n]!=0)
                                printf("q%d,",n);
                }
                printf("}");
        }
    }
    printf("\n Final states:");
    findfinalstate();




}

void findclosure(int x,int sta)
{
        struct node *temp;
        int i;
       if(buffer[x])
```

```
                return;
          e_closure[sta][c++]=x;
         buffer[x]=1;
          if(alphabet[noalpha-1]=='e' && transition[x][noalpha-1]!=NULL)
            {
                    temp=transition[x][noalpha-1];
                    while(temp!=NULL)
                   {
                            findclosure(temp->st,sta);
                            temp=temp->link;
                    }
            }
 }

void insert_trantbl(int r,char c,int s)
{
         int j;
         struct node *temp;
          j=findalpha(c);
        if(j==999)
        {
                 printf("error\n");
                exit(0);
        }
       temp=(struct node *) malloc(sizeof(struct node));
       temp->st=s;
       temp->link=transition[r][j];
```

```
        transition[r][j]=temp;
}

int findalpha(char c)
{
        int i;
        for(i=0;i<noalpha;i++)
            if(alphabet[i]==c)
                return i;

          return(999);


}

void unionclosure(int i)
{
         int j=0,k;
        while(e_closure[i][j]!=0)
        {
                k=e_closure[i][j];
                set[k]=1;
                j++;
        }
}
void findfinalstate()
{
```

```
        int i,j,k,t;
        for(i=0;i<nofinal;i++)
        {
                for(j=1;j<=nostate;j++)
                {
                        for(k=0;e_closure[j][k]!=0;k++)
                         {
                                if(e_closure[j][k]==finalstate[i])
                                {

                                        print_e_closure(j);
                                }
                         }
                }
         }


 }

void print_e_closure(int i)
{
       int j;
       printf("{");
       for(j=0;e_closure[i][j]!=0;j++)
                    printf("q%d,",e_closure[i][j]);
        printf("}\t");
}
```

Output

```
D:\CD Lab>a
enter the number of alphabets?
4
NOTE:- [ use letter e as epsilon]
NOTE:- [e must be last character ,if it is present]

Enter alphabets?
a b c e
Enter the number of states?
3
Enter the start state?
1
Enter the number of final states?
1
Enter the final states?
3
Enter no of transition?
5
NOTE:- [Transition is in the form--> qno   alphabet   qno]
NOTE:- [States number must be greater than zero]

Enter transition?
1 a 1
1 e 2
2 b 2
2 e 3
3 c 3

Equivalent NFA without epsilon
-----------------------------------
start state:{q1,q2,q3,}
Alphabets:a b c e
 States :{q1,q2,q3,}    {q2,q3,}        {q3,}
Tnransitions are...:

{q1,q2,q3,}     a       {q1,q2,q3,}
{q1,q2,q3,}     b       {q2,q3,}
{q1,q2,q3,}     c       {q3,}
{q2,q3,}        a       {}
{q2,q3,}        b       {q2,q3,}
{q2,q3,}        c       {q3,}
{q3,}   a       {}
{q3,}   b       {}
{q3,}   c       {q3,}
 Final states:{q1,q2,q3,}       {q2,q3,}        {q3,}
```

12.Write program to convert NFA to DFA

Program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
 int st;
 struct node *link;
};
struct node1
{

 int nst[20];
};

void insert(int ,char, int);
int findalpha(char);
void findfinalstate(void);
int insertdfastate(struct node1);
int compare(struct node1,struct node1);
void printnewstate(struct node1);
static                                                                    int
set[20],nostate,noalpha,s,notransition,nofinal,start,finalstate[20],c,r,buffer[20
];
int complete=-1;
char alphabet[20];
static int eclosure[20][20]={0};
struct node1 hash[20];
struct node * transition[20][20]={NULL};
```

```
void main()
{
 int i,j,k,m,t,n,l;
 struct node *temp;
 struct node1 newstate={0},tmpstate={0};

 printf("Enter the number of alphabets?\n");
 printf("NOTE:- [ use letter e as epsilon]\n");
 printf("NOTE:- [e must be last character ,if it is present]\n");
 printf("\nEnter No of alphabets and alphabets?\n");
 scanf("%d",&noalpha);
 getchar();
 for(i=0;i<noalpha;i++)
 {

 alphabet[i]=getchar();
 getchar();
 }
 printf("Enter the number of states?\n");
 scanf("%d",&nostate);
 printf("Enter the start state?\n");
 scanf("%d",&start);
 printf("Enter the number of final states?\n");
 scanf("%d",&nofinal);
 printf("Enter the final states?\n");
 for(i=0;i<nofinal;i++)
 scanf("%d",&finalstate[i]);
 printf("Enter no of transition?\n");

 scanf("%d",&notransition);
```

```
   printf("NOTE:- [Transition is in the form–> qno alphabet qno]\n",notransition);
 printf("NOTE:- [States number must be greater than zero]\n");
 printf("\nEnter transition?\n");


 for(i=0;i<notransition;i++)
 {


  scanf("%d %c%d",&r,&c,&s);
  insert(r,c,s);

 }
 for(i=0;i<20;i++)
 {
  for(j=0;j<20;j++)
  hash[i].nst[j]=0;
 }
 complete=-1;
 i=-1;
 printf("\nEquivalent DFA.....\n");
 printf(".....\n");

 printf("Trnsitions of DFA\n");

 newstate.nst[start]=start;
 insertdfastate(newstate);
 while(i!=complete)
 {
```

```
i++;
newstate=hash[i];
for(k=0;k<noalpha;k++)
{
 c=0;
 for(j=1;j<=nostate;j++)
 set[j]=0;
 for(j=1;j<=nostate;j++)
 {
  l=newstate.nst[j];
  if(l!=0)
  {
   temp=transition[l][k];
   while(temp!=NULL)
   {
    if(set[temp->st]==0)
    {
     c++;
     set[temp->st]=temp->st;
    }
    temp=temp->link;


   }
  }
 }
printf("\n");
if(c!=0)
{
 for(m=1;m<=nostate;m++)
```

```
     tmpstate.nst[m]=set[m];

    insertdfastate(tmpstate);

    printnewstate(newstate);
    printf("%c\t",alphabet[k]);
    printnewstate(tmpstate);
    printf("\n");
    }
    else
    {
    printnewstate(newstate);
    printf("%c\t", alphabet[k]);
    printf("NULL\n");
    }

  }
   }
printf("\nStates of DFA:\n");
for(i=0;i<=complete;i++)
printnewstate(hash[i]);
printf("\n Alphabets:\n");
for(i=0;i<noalpha;i++)
printf("%c\t",alphabet[i]);
printf("\n Start State:\n");
 printf("q%d",start);
printf("\nFinal states:\n");
findfinalstate();

}
```

```
int insertdfastate(struct node1 newstate)
{
 int i;
 for(i=0;i<=complete;i++)
 {
  if(compare(hash[i],newstate))
   return 0;
 }
 complete++;
 hash[complete]=newstate;
 return 1;
}
int compare(struct node1 a,struct node1 b)
{
 int i;


  for(i=1;i<=nostate;i++)
  {
   if(a.nst[i]!=b.nst[i])
    return 0;
  }
  return 1;


}

void insert(int r,char c,int s)
{
      int j;
```

```
        struct node *temp;
        j=findalpha(c);
        if(j==999)
        {
  printf("error\n");
  exit(0);
        }
        temp=(struct node *) malloc(sizeof(struct node));
        temp->st=s;
        temp->link=transition[r][j];
        transition[r][j]=temp;
}

int findalpha(char c)
{
 int i;
 for(i=0;i<noalpha;i++)
 if(alphabet[i]==c)
  return i;

  return(999);


}


void findfinalstate()
{
 int i,j,k,t;
```

```
  for(i=0;i<=complete;i++)
  {
  for(j=1;j<=nostate;j++)
  {
   for(k=0;k<nofinal;k++)
   {
    if(hash[i].nst[j]==finalstate[k])
    {
     printnewstate(hash[i]);
     printf("\t");
     j=nostate;
     break;
    }
   }
  }
  }
}


void printnewstate(struct node1 state)
{
 int j;
 printf("{");
  for(j=1;j<=nostate;j++)
  {
   if(state.nst[j]!=0)
    printf("q%d,",state.nst[j]);
  }
  printf("}\t");
```

}

Output

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19042.1288]
(c) Microsoft Corporation. All rights reserved.

D:\CD Lab>gcc nfatodfa.c

D:\CD Lab>a
Enter the number of alphabets?
NOTE:- [ use letter e as epsilon]
NOTE:- [e must be last character ,if it is present]

Enter No of alphabets and alphabets?
2
a b
Enter the number of states?
4
Enter the start state?
1
Enter the number of final states?
2
Enter the final states?
3 4
Enter no of transition?
8
NOTE:- [Transition is in the formΓÇô> qno alphabet qno]
NOTE:- [States number must be greater than zero]

Enter transition?
1 a 1
1 b 1
1 a 2
2 b 2
2 a 3
3 a 4
3 b 4
4 b 3
```

```
C:\Windows\System32\cmd.exe
Equivalent DFA.....
.....
Trnsitions of DFA

{q1,}   a       {q1,q2,}

{q1,}   b       {q1,}

{q1,q2,}        a       {q1,q2,q3,}

{q1,q2,}        b       {q1,q2,}

{q1,q2,q3,}     a       {q1,q2,q3,q4,}

{q1,q2,q3,}     b       {q1,q2,q4,}

{q1,q2,q3,q4,}  a       {q1,q2,q3,q4,}

{q1,q2,q3,q4,}  b       {q1,q2,q3,q4,}

{q1,q2,q4,}     a       {q1,q2,q3,}

{q1,q2,q4,}     b       {q1,q2,q3,}

States of DFA:
{q1,}   {q1,q2,}        {q1,q2,q3,}     {q1,q2,q3,q4,}  {q1,q2,q4,}
 Alphabets:
a       b
 Start State:
q1
Final states:
{q1,q2,q3,}             {q1,q2,q3,q4,}          {q1,q2,q4,}
D:\CD Lab>
```

13.Write program to minimize any given DFA.

Program

```
#include <stdio.h>
#include <stdlib.h>
static int nostate,noalpha,s,notransition,nofinal,start,finalstate[20],r;
char alphabet[20];
int transition_map[30][30], table[30][30], nonfinalstate[20],
partition[20][20];
int findalpha(char a)
{
 int i;
 for(i=0;i<noalpha;i++)
 if(alphabet[i]==a)
  return i;
 return(-1);
}
int main() {
 int i,j,p[20],q[20],k;
 char a;
 for(i=0;i<30;i++){
  for(j=0;j<30;j++)
          transition_map[i][j]=-1;
 }
 printf("Enter the number of alphabets: ");
 scanf("%d",&noalpha);
 getchar();
 printf("Enter the alphabets: \n");
```

```
for(i=0;i<noalpha;i++)
{
 alphabet[i]=getchar();
 getchar();
}
printf("Enter the number of states: ");
scanf("%d",&nostate);
printf("Enter the start state: ");
scanf("%d",&start);
printf("Enter the number of final states: ");
scanf("%d",&nofinal);
printf("Enter the final states:\n");
for(i=0;i<nofinal;i++)
  scanf("%d",&finalstate[i]);
printf("Enter no of transition: ");
scanf("%d",&notransition);
printf("Enter Transition in the form –> state alphabet next_state\n");
for(i=0;i<notransition;i++)
{
 scanf("%d %c %d",&r,&a,&s);
 j=findalpha(a);
 if (j==-1){printf("\nerror\n"); exit(1);}
 transition_map[r][j] = s;
}
for(i=0;i<nostate;i++){
 for(j=0;j<i;j++){
          table[i][j]=0;
```

```
 }
}
int f=0;
k=0;
for(i=0;i<nostate;i++){
 f=0;
 for(j=0;j<nofinal;j++){
          if(i==finalstate[j])
          { f=1;break;}
 }
 if(f==0){nonfinalstate[k++]=i;}
}
for(i=0;i<nofinal;i++){
for(j=0;j<(nostate-nofinal);j++)
     if(nonfinalstate[j]>finalstate[i])
    table[nonfinalstate[j]][finalstate[i]]=1;
     else
    table[finalstate[i]][nonfinalstate[j]]=1;
}
int change = 1;
while(change==1){
 change=0;
 for(i=0;i<nostate;i++){
   for(j=0;j<i;j++){
          if(table[i][j]!=1){
                  for(k=0;k<noalpha;k++)
                          p[k]=transition_map[i][k];
```

```
                for(k=0;k<noalpha;k++)
                        q[k]=transition_map[j][k];
                for(k=0;k<noalpha;k++){
                        if(p[k]>q[k]){
                                if (table[p[k]][q[k]]==1){
                                        change=1;
                                        table[i][j]=1;
                                        break;
                                }
                        }
                        else if(p[k]<q[k]){
                                if (table[q[k]][p[k]]==1){
                                        change=1;
                                        table[i][j]=1;
                                        break;
                                }
                        }
                }
        }
    }
    }
}
k=0;
for(i=0;i<nostate;i++){
 k=0;
 partition[i][k++]=i;
 for(j=0;j<i;j++)
```

```
            if(table[i][j]==0){
                  partition[i][k++]=j;
            }
 partition[i][k]=-1;
}
int newstate[20]={0},m;

printf("\nStates in minimized DFA");
printf("\n---------------------------\n");
for(i=nostate-1;i>=0;i--){
 k=0;
 if(newstate[i]==0){
 printf("{");
 while(partition[i][k]!=-1){
            if(newstate[partition[i][k]]==0){
                  newstate[partition[i][k]]=1;
                  printf("q%d ",partition[i][k]);
            }
            k++;
 }
 printf("}\n");
}
}
return 0;
}
```

Output

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19042.1288]
(c) Microsoft Corporation. All rights reserved.

D:\CD Lab>gcc 10_1.4_CODE.c

D:\CD Lab>a
Enter the number of alphabets: 2
Enter the alphabets:
0
1
Enter the number of states: 5
Enter the start state: 0
Enter the number of final states: 1
Enter the final states:
4
Enter no of transition: 10
Enter Transition in the form ΓÇô> state alphabet next_state
0 0 1
0 1 2
1 0 1
1 1 3
2 0 1
2 1 2
3 0 1
3 1 4
4 0 1
4 1 2

States in minimized DFA
---------------------------
{q4 }
{q3 }
{q2 q0 }
{q1 }
```

14. Write program to find Simulate First and Follow of any given grammar

Program

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

int n;
char prods[50][50];
char firsts[26][50];
int is_first_done[26];

char follows[26][50];
int is_follow_done[26];

int isTerminal(char c)
{
   if (c < 65 || c > 90)
      return 1;
   return 0;
}

void first(char nonterm)
{
   int index = 0;
   char curr_firsts[50];
   for (int i = 0; i < n; i++)
   {
      if (prods[i][0] == nonterm)
```

```
{

   int curr_prod_index = 2;

   int flag = 0;
   while (prods[i][curr_prod_index] != '\0' && flag == 0)
   {
      flag = 1;
      if (isTerminal(prods[i][curr_prod_index]))
      {
         curr_firsts[index] = prods[i][2];
         index++;
         break;
      }

      if (!is_first_done[prods[i][curr_prod_index] - 65])
         first(prods[i][curr_prod_index]);

      int in = 0;
      while (firsts[prods[i][curr_prod_index] - 65][in] != '\0')
      {
         curr_firsts[index] = firsts[prods[i][curr_prod_index] - 65][in];
         if (firsts[prods[i][curr_prod_index] - 65][in] == 'e')
         {
            curr_prod_index++;
            flag = 0;
         }
```

```
                index++;
                in++;
            }
        }
    }

    curr_firsts[index] = '\0';
    index++;
    strcpy(firsts[nonterm - 65], curr_firsts);
    is_first_done[nonterm - 65] = 1;
}

void follow(char nonterm)
{
    int index = 0;
    char curr_follows[50];
    if (nonterm == prods[0][0])
    {
        curr_follows[index] = '$';
        index++;
    }
    for (int j = 0; j < n; j++)
    {
        int k = 2;
        int include_lhs_flag;
        while (prods[j][k] != '\0')
```

```
{
   include_lhs_flag = 0;
   if (prods[j][k] == nonterm)
   {
      if (prods[j][k + 1] != '\0')
      {
         if (isTerminal(prods[j][k + 1]))
         {
            curr_follows[index] = prods[j][k + 1];
            index++;
            break;
         }

         int in = 0;
         while (firsts[prods[j][k + 1] - 65][in] != '\0')
         {
            if (firsts[prods[j][k + 1] - 65][in] == 'e')
            {
               include_lhs_flag = 1;
               in++;
               continue;
            }

            int temp_flag = 0;
            for (int z = 0; z < index; z++)
               if (firsts[prods[j][k + 1] - 65][in] == curr_follows[z])
               {
```

```
                temp_flag = 1;
                in++;
                break;
            }
        if (temp_flag)
            continue;

        curr_follows[index] = firsts[prods[j][k + 1] - 65][in];
        index++;
        in++;
    }
}

if (prods[j][k + 1] == '\0' || include_lhs_flag == 1)
{
    if (prods[j][0] != nonterm)
    {
        if (!is_follow_done[prods[j][0] - 65])
            follow(prods[j][0]);
        int x = 0;
        while (follows[prods[j][0] - 65][x] != '\0')
        {
            int temp_flag = 0;
            for (int z = 0; z < index; z++)
                if (follows[prods[j][0] - 65][x] == curr_follows[z])
                {
                    temp_flag = 1;
```

```
                        x++;
                        break;
                    }
                if (temp_flag)
                    continue;
                curr_follows[index] = follows[prods[j][0] - 65][x];
                index++;
                x++;
            }
        }
    }
    k++;
  }
}

curr_follows[index] = '\0';
index++;
strcpy(follows[nonterm - 65], curr_follows);
is_follow_done[nonterm - 65] = 1;
}

int main()
{
printf("Enter the number of productions\n");
scanf("%d", &n);
printf("Enter productions: \n");
```

```
    for (int i = 0; i < n; i++)
        scanf("%s", prods[i]);

    for (int i = 0; i < 26; i++)
        is_first_done[i] = 0;


    for (int i = 0; i < n; i++)
        if (is_first_done[prods[i][0] - 65] == 0)
            first(prods[i][0]);

    for (int i = 0; i < n; i++)
        if (is_follow_done[prods[i][0] - 65] == 0)
            follow(prods[i][0]);

    printf("Firsts:\n");
    for (int i = 0; i < 26; i++)
        if (is_first_done[i])
            printf("%c : %s\n", i + 65, firsts[i]);

    printf("Follows:\n");
    for (int i = 0; i < 26; i++)
        if (is_follow_done[i])
            printf("%c : %s\n", i + 65, follows[i]);
}
```

Output

```
mec@cc-2-1: ~/CSB10
File  Edit  View  Search  Terminal  Help
Enter the number of productions
8
Enter productions:
E=TR
R=+TR
R=e
T=FY
Y=*FY
Y=e
F=(E)
F=i
Firsts:
E : (i
F : (i
R : +e
T : (i
Y : *e
Follows:
E : $)
F : *+$)
R : $)
T : +$)
Y : +$)
mec@cc-2-1:~/CSB10$
```

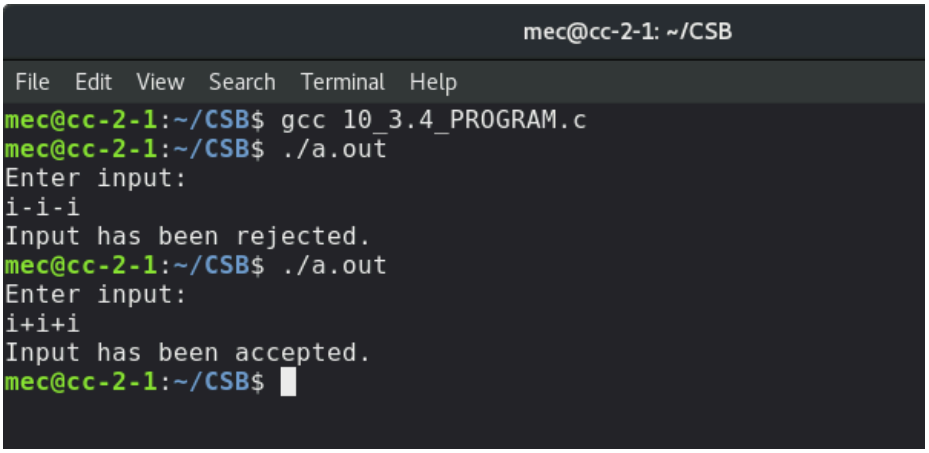
15. Design and implement a recursive descent parser for a given grammar.

Program

```
#include <stdio.h>

char inp[100];
int len = 0;
int curr = 0;

int E();
int Z();

int main()
{
   printf("Enter input:\n");
   scanf("%s", inp);
   while (inp[len] != '\0')
      len++;
   int res = E();
   if (res == 1 && curr == len)
      printf("Input has been accepted.\n");
   else
      printf("Input has been rejected.\n");
}

int E()
{
   int result;
```

```
    if (inp[curr] == 'i')
    {
        curr++;
        result = Z();
        if (result == 1)
            return 1;
        else
            return -1;
    }
    return -1;
}

int Z()
{
    int result;
    if (inp[curr] == '+' && inp[curr + 1] == 'i')
    {
        curr += 2;
        result = Z();
        if (result == 1)
            return 1;
    }
    return 1;
}
```

Output


mec@cc-2-1: ~/CSB
File Edit View Search Terminal Help
mec@cc-2-1:~/CSB$ gcc 10_3.4_PROGRAM.c
mec@cc-2-1:~/CSB$ ./a.out
Enter input:
i-i-i
Input has been rejected.
mec@cc-2-1:~/CSB$ ./a.out
Enter input:
i+i+i
Input has been accepted.
mec@cc-2-1:~/CSB$

16. Construct a Shift Reduce Parser for a given language.

Program

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

char inp[100];
int len;
char stack[100];
int top = 0;

void print_details(int ind, char *action)
{
   printf("$");
   for (int i = 0; i <= top; i++)
      printf("%c", stack[i]);
   printf("\t\t");
   for (int i = ind; i < len; i++)
      printf("%c", inp[i]);
   printf("$\t\t%s\n", action);
}
void check_for_reduce(int i)
{
   int flag = 1;
   while (flag)
   {
      flag = 0;
      if (stack[top - 2] == 'S' && stack[top - 1] == '+' && stack[top] == 'S')
```

```
        {
           print_details(i + 1, "REDUCE");
           stack[top - 2] = 'S';
           top = top - 2;
           flag = 1;
        }
         else if (stack[top - 2] == 'S' && stack[top - 1] == '-' && stack[top] ==
'S')
        {
           print_details(i + 1, "REDUCE");
           stack[top - 2] = 'S';
           top = top - 2;
           flag = 1;
        }
         else if (stack[top - 2] == '(' && stack[top - 1] == 'S' && stack[top] ==
')')
        {
           print_details(i + 1, "REDUCE");
           stack[top - 2] = 'S';
           top = top - 2;
           flag = 1;
        }
        else if (stack[top] == 'i')
        {
           print_details(i + 1, "REDUCE");
           stack[top] = 'S';
           flag = 1;
```

```
        }
    }
}
int main()
{
    printf("Enter input:\n");
    scanf("%s", inp);
    len = strlen(inp);

    printf("Stack\t\tInput\t\tAction\n");

    for (int i = 0; i < len; i++)
    {
        print_details(i, "SHIFT");
        stack[top] = inp[i];
        check_for_reduce(i);
        top++;
    }
    top--;

    if (top == 0 && stack[0] == 'S')
        printf("Accepted.\n");
    else
        printf("Rejected.\n");
}
```

Output

```
File  Edit  View  Search  Terminal  Help
mec@cc-2-1:~$ cd CSB10
mec@cc-2-1:~/CSB10$ gcc 10_3.3_PROGRAM.c
mec@cc-2-1:~/CSB10$ ./a.out
Enter input:
i-(i+i)
Stack           Input           Action
$               i-(i+i)$                SHIFT
$i              -(i+i)$         REDUCE
$S              -(i+i)$         SHIFT
$S-             (i+i)$          SHIFT
$S-(            i+i)$           SHIFT
$S-(i           +i)$            REDUCE
$S-(S           +i)$            SHIFT
$S-(S+          i)$             SHIFT
$S-(S+i         )$              REDUCE
$S-(S+S         )$              REDUCE
$S-(S+          )$              SHIFT
$S-(S)          $               REDUCE
$S-S            $               REDUCE
Accepted.
mec@cc-2-1:~/CSB10$ gcc 10_3.3_PROGRAM.c
mec@cc-2-1:~/CSB10$ ./a.out
Enter input:
i+i*i
Stack           Input           Action
$               i+i*i$          SHIFT
$i              +i*i$           REDUCE
$S              +i*i$           SHIFT
$S+             i*i$            SHIFT
$S+i            *i$             REDUCE
$S+S            *i$             REDUCE
$S+             *i$             SHIFT
$S*S            i$              SHIFT
$S*i            $               REDUCE
Rejected.
mec@cc-2-1:~/CSB10$
```

17. Write a program to perform constant propagation.

PROGRAM CODE:

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
#include<conio.h>
void input();
void output();
void change(int p,char *res);
void constant();
struct expr
{
char op[2],op1[5],op2[5],res[5];
int flag;
}arr[10];
int n;
void main()
{
clrscr();
input();
constant();
output();
getch();
}
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]) || strcmp(arr[i].op,"=")==0) /*if both digits,
store them in variables*/
{
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;
case '=':
res=op1;
break;
}
sprintf(res1,"%d",res);
arr[i].flag=1; /*eliminate expr and replace any operand below that uses result of this expr */
change(i,res1);
}
}
}
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",arr[i].op,arr[i].op1,arr[i].op2,arr[i].res);
}
}
}
void change(int p,char *res)
{
int i;
for(i=p+1;i<n;i++)
```

```
{
if(strcmp(arr[p].res,arr[i].op1)==0)
strcpy(arr[i].op1,res);
else if(strcmp(arr[p].res,arr[i].op2)==0)
strcpy(arr[i].op2,res);
}
}
```

OUTPUT:

```
Enter the maximum number of  expressions : 4

Enter the input :
= 3 - a
+ a b t1
+ a c t2
+ t1 t2 t3

Optimized code is :
+ 3 b t1
+ 3 c t2
```

18. Implement Intermediate code generation for simple expressions.

Program

```
#include <stdio.h>
#include <string.h>
void gen_code_for_operator(char *inp, char operator, char * reg)
{    int i = 0, j = 0; // j is used as an index of temp, i is used as an index of inp
   char temp[100];
   while (inp[i] != '\0')
   {
      if (inp[i] == operator)
      {
         printf("%c\t%c\t%c\t%c\n", operator, * reg, inp[i - 1], inp[i + 1]);
           temp[j - 1] = *reg; // Instead of copying a/b to the temp string, copy
the output register Z
         i += 2;
         (*reg)--; // Change register from Z to Y etc
         continue;
      }
      temp[j] = inp[i];
      i++;
      j++;
   }
   temp[++j] = '\0';
   strcpy(inp, temp);
}
```

```
void gen_code(char *inp)
{
   // Operator precendence - /, *, +, -, =
   char reg = 'Z'; // Decremented to get Z, Y etc
   gen_code_for_operator(inp, '/', &reg);
   gen_code_for_operator(inp, '*', &reg);
   gen_code_for_operator(inp, '+', &reg);
   gen_code_for_operator(inp, '-', &reg);
   gen_code_for_operator(inp, '=', &reg);
}
int main()
{
   char inp[100];
   printf("Enter expression:\n\n");
   scanf("%s", inp);
   printf("Oprtr\tDestn\tOp1\tOp2\n");
   gen_code(inp);
}
```

Output

```
mec@cl1-3-1: ~/Documents/CSB
File  Edit  View  Search  Terminal  Help
mec@cl1-3-1:~/Documents/CSB$ gcc 10_4.1_PROGRAM.c
mec@cl1-3-1:~/Documents/CSB$ ./a.out
Enter expression:

q=a+b-c/d*2
Oprtr   Destn   Op1     Op2
/       Z       c       d
*       Y       Z       2
+       X       a       b
-       W       X       Y
=       V       q       W
mec@cl1-3-1:~/Documents/CSB$
```

19. 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 an 8086 assembler. The target assembly instructions can be simple move, add, sub, jump etc

Program

```
#include <stdio.h>
#include <stdio.h>
//#include<conio.h>
#include <string.h>
void main()
{
   char icode[10][30], str[20], opr[10];
   int i = 0;
   //clrscr();
   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);
    //getch();
}
```

Output

```
mec@cl1-3-1: ~/Documents/CSB
File  Edit  View  Search  Terminal  Help
mec@cl1-3-1:~/Documents/CSB$ gcc 10_4.2_PROGRAM.c
mec@cl1-3-1:~/Documents/CSB$ ./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
        MULb,R0
        Mov R0,a
        Mov f,R1
        MULh,R1
        Mov R1,c
        Mov a,R2
        MULh,R2
        Mov R2,g
        Mov Q,R3
        ADDw,R3
        Mov R3,f
        Mov q,R4
        SUBj,R4
mec@cl1-3-1:~/Documents/CSB$
```