College of Engineering Trivandrum

Compiler Design Lab
Final Report

Abhishek Manoharan
S7 CSE Roll No:2

TVE17CS002

Supervisors: Remya Krishnan, Thara R J, Thania Kumar

A report submitted in partial fulfilment of Compiler Lab final exam
in Semester 7

Department of Computer Science

January 13, 2021

Contents

1 Lexical analyzer **4**
1.1 Aim 4
1.2 Theory 4
1.3 Algorithm 4
1.4 Code 4
1.5 Output 8
1.6 Result 9

2 Lexical analyzer **10**
2.1 Aim 10
2.2 Theory 10
2.2.1 Program Structure: 10
2.3 Algorithm 11
2.4 Code 11
2.5 Output 12
2.6 Result 13

3 YACC **14**
3.1 Aim 14
3.2 Theory 14
3.3 Algorithm 14
3.4 Arithmetic Expression 14
3.4.1 Question 14
3.4.2 Algorithm 14
3.4.3 Code 14
3.4.4 Output 16
3.5 Identifier 17
3.5.1 Question 17
3.5.2 Algorithm 17
3.5.3 Code 17
3.5.4 Output 18
3.6 Calculator 19
3.6.1 Question 19
3.6.2 Algorithm 19
3.6.3 Code 19
3.6.4 Output 21
3.7 BNF to YACC 22
3.7.1 Question 22
3.7.2 Algorithm 22
3.7.3 Code 22
3.7.4 Output 28
3.8 Result 28

4 ϵ- closure **29**
4.1 Aim 29
4.2 Theory 29
4.3 Algorithm 29
4.4 Code 30
4.5 Output 31
4.6 Result 31

5 ϵ- NFA to NFA **32**
5.1 Aim 32
5.2 Theory 32
5.3 Algorithm 32
5.4 Code 33
5.5 Output 36
5.6 Result 37

6 NFA to DFA **38**
6.1 Aim 38
6.2 Theory 38
6.3 Algorithm 39
6.4 Code 40
6.5 Output 42
6.6 Result 42

7 DFA Minimisation **43**
7.1 Aim 43
7.2 Theory 43
7.3 Algorithm 44
7.4 Code 44
7.5 Output 47
7.6 Result 48

8 Operator Precedence Parsing **49**
8.1 Aim 49
8.2 Theory 49
8.3 Algorithm 49
8.4 Code 49
8.5 Output 51
8.6 Result 52

9 First and Follow **53**
9.1 Aim 53
9.2 Theory 53
9.3 Algorithm 53
9.4 Code 53
9.5 Output 58
9.6 Result 59

10 Recursive descent parser **60**
10.1 Aim 60
10.2 Theory 60
10.3 Algorithm 60
10.4 Code 60
10.5 Output 63
10.6 Result 65

11 Shift Reduce Parser **66**
11.1 Aim 66
11.2 Theory 66
11.3 Algorithm 66
11.4 Code 67
11.5 Output 68
11.6 Result 70

12 Loop Unrolling **71**
12.1 Aim 71
12.2 Theory 71
12.2.1 Advantages 71
12.2.2 Disadvantages 71
12.3 Algorithm 71
12.4 Code 71
12.5 Output 74
12.6 Result 76

13 Constant Propagation 77
13.1 Aim . . . 77
13.2 Theory . . . 77
13.3 Algorithm . . . 77
13.4 Code . . . 77
13.5 Output . . . 79
13.6 Result . . . 80

14 Intermediate Code Generation 81
14.1 Aim . . . 81
14.2 Theory . . . 81
14.3 Algorithm . . . 82
14.4 Code . . . 82
14.5 Output . . . 84
14.6 Result . . . 85

Exp 1

1 Lexical analyzer

1.1 Aim

Design and Implement a lexical analyzer for given language using C and the lexical analyzer should ignore redundant spaces, tabs and new line.

1.2 Theory

The very first phase of a compiler deals with lexical analysis. A lexical analyser, also known as scanner, converts the high level input program into a sequence of tokens. A lexical token is a sequence of characters which is treated as a unit in the grammar of the programming languages.

The common type of tokens include:
Keyword: A keyword is a word reserved by a programming language having a special meaning.
Identifier: It is a user-defined name used to uniquely identify a program element. It can be a class, method, variable, namespace etc.
Operator: It is a symbol that tells the compiler or interpreter to perform specific mathematical, relational or logical operation and produce final result.
Separator: Separators are used to separate one programming element from the other.
Literals: A literal is a notation for representing a fixed value and do not change during the course of execution of the program.

1.3 Algorithm

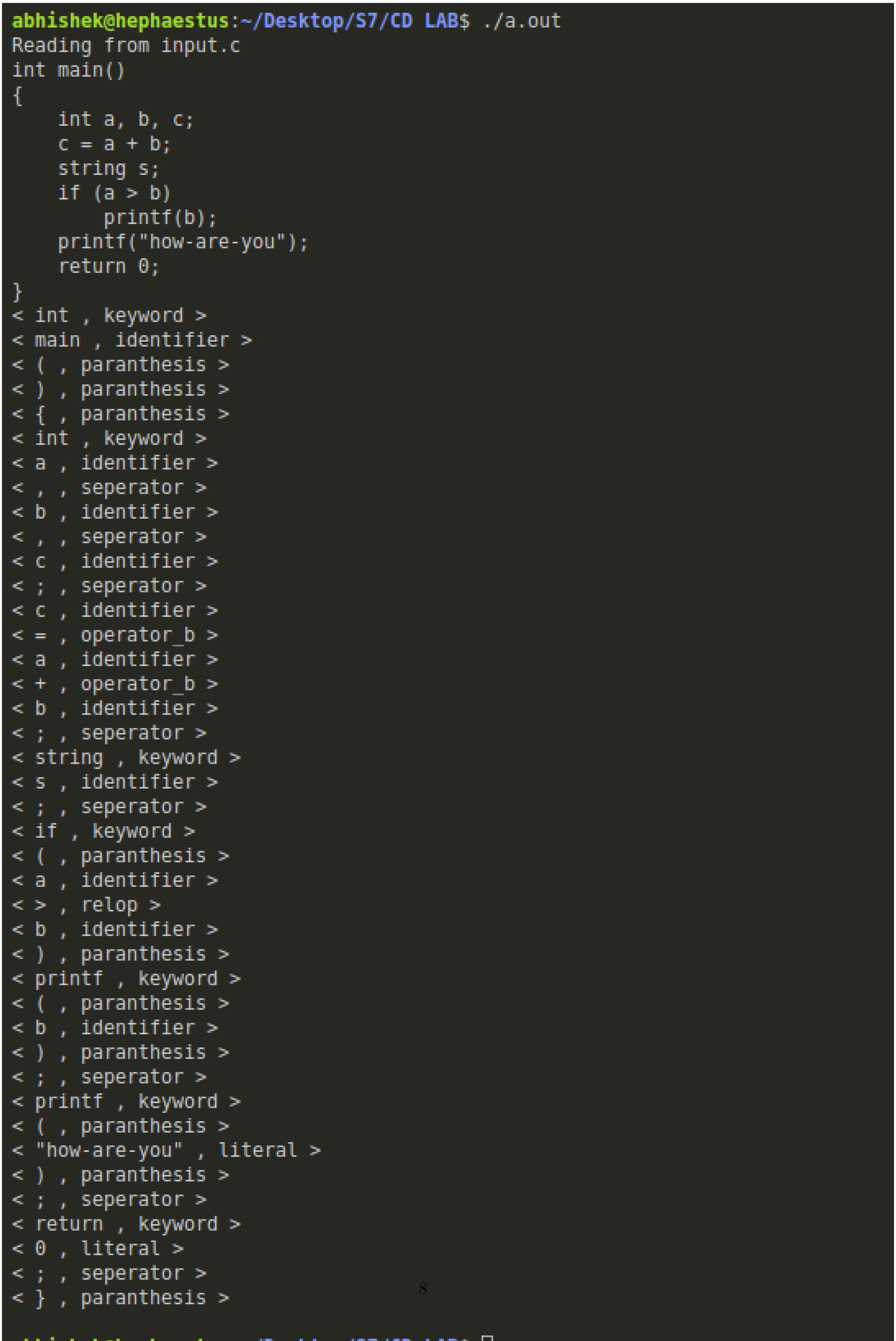
Algorithm 1: Lexical Analyser Algorithm

```
START
Get the input file and read from the file word by word .
Split the word into meaningful tokens with the help of delimiters
Read each token one by one
      If token is a keyword
         print < token , keyword >
      If token is an operator
         print < token , operator >
      If token is a separator / delimiter
         print < token , delimiter >
      If token is a literal
         print < token , literal >
      If token is an identifier
         print < token , identifier >
STOP
```

1.4 Code

```
#include <bits/stdc++.h>
using namespace std;
vector<string> split_vect(vector<string> s)
{
    vector<string> words;
    for (auto x : s)
    {
        int n = x.size();
        int j = 0;
        for (int i = 0; i < n; ++i)
        {
            if (x[i] == ’ ’ || x[i] == ’\t’)
            {
                if (i != j)
                {
                    words.push_back(x.substr(j, i - j));
                    j = i + 1;
                }
                else
                {
                    j++;
                }
            }
            if (x[i] == '{' || x[i] == '}' || x[i] == '(' || x[i] == ')' || x[i] == ',' || x[i] == ';')
            {
                if (i != j)
                {
                    words.push_back(x.substr(j, i - j));
                    j = i;
                }
                string samp = "";
                samp += x[i];

                if (i + 1 != n)
                {
                    words.push_back(samp);
                    j = i + 1;
                }
            }
        }
        words.push_back(x.substr(j, n - j));
    }
    return words;
}

bool is_key(string s)
{
    if (s == "if" || s == "else" || s == "int" || s == "for" || s == "bool" || s == "string" || s == "float" || s == "return" || s == "printf")
        return true;
    else
        return false;
}

bool is_id(string s)
{
    int size = s.size();
    if (!isalpha(s[0]))
    {
        return false;
    }
    else
    {
        for (int i = 1; i < size; ++i)
        {
            if (!isalnum(s[i]))
            {
                return false;
            }
        }
    }
    return true;
}

bool is_bop(string s)
{
    if (s == "+" || s == "-" || s == "*" || s == "/" || s == "&&" || s == "||" || s == "=")
    {
        return true;
    }
    else
    {
        return false;
    }
}

bool is_uop(string s)
{
    if (s == "++" || s == "--" || s == "!")
    {
        return true;
    }
    else
    {
        return false;
    }
}

bool is_par(string s)
{
    if (s == "{" || s == "}" || s == "(" || s == ")")
    {
        return true;
    }
    return false;
}

bool is_relop(string s)
{
    if (s == " < " || s == ">" || s == ">=" || s == "<=" || s == "==")
    {
        return true;
    }
    return false;
}

bool is_num(string s)
{
    int n = s.size();
    if (n == 0)
        return false;
    if (s[0] == ’"’ && s[n - 1] == ’"’ && n - 1 != 0)
        return true;
    for (int i = 0; i < n; ++i)
    {
        if (!isdigit(s[i]))
        {
            if (s[i] != ’.’)
                return false;
            else
            {
                for (int j = i + 1; j < n; ++j)
                {
                    if (!isdigit(s[j]))
                    {
                        return false;
                    }
                }
                return true;
            }
        }
    }
    return true;
}

bool is_sup(string s)
{
    if (s == "," || s == ";")
        return true;
    return false;
}
int main()
{
    vector<string> lines;
    string s;
    ifstream file("input.c");
    cout << "Reading from input.c" << endl;
    while (getline(file, s))
    {
        cout << s << endl;
        lines.push_back(s);
    }
    vector<string> words;
    words = split_vect(lines);
    vector<string> tokens;
    for (auto x : words)
    {
        if (is_key(x))
        {
            tokens.push_back("< " + x + " , " + "keyword" + " >");
        }
        else if (is_id(x))
        {
            tokens.push_back("< " + x + " , " + "identifier" + " >");
        }
        else if (is_par(x))
        {
            tokens.push_back("< " + x + " , " + "paranthesis" + " >");
        }
        else if (is_bop(x))
        {
            tokens.push_back("< " + x + " , " + "operator_b" + " >");
        }
        else if (is_uop(x))
        {
            tokens.push_back("< " + x + " , " + "operator_u" + " >");
        }
        else if (is_relop(x))
        {
            tokens.push_back("< " + x + " , " + "relop" + " >");
        }
        else if (is_sup(x))
        {
            tokens.push_back("< " + x + " , " + "seperator" + " >");
        }
        else if (is_num(x))
        {
            tokens.push_back("< " + x + " , " + "literal" + " >");
        }
        else
        {
            tokens.push_back("< " + x + " , " + "no_idea" + " >");
            cout << "un identified tocken " << x << " program forced to quit" << endl;
            return 0;
        }
    }
    for (auto x : tokens)
    {
        cout << x << endl;
        ;
    }
    cout << endl;
    return 0;
}
```

1.5 Output



```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Reading from input.c
int main()
{
    int a, b, c;
    c = a + b;
    string s;
    if (a > b)
        printf(b);
    printf("how-are-you");
    return 0;
}
< int , keyword >
< main , identifier >
< ( , paranthesis >
< ) , paranthesis >
< { , paranthesis >
< int , keyword >
< a , identifier >
< , , seperator >
< b , identifier >
< , , seperator >
< c , identifier >
< ; , seperator >
< c , identifier >
< = , operator_b >
< a , identifier >
< + , operator_b >
< b , identifier >
< ; , seperator >
< string , keyword >
< s , identifier >
< ; , seperator >
< if , keyword >
< ( , paranthesis >
< a , identifier >
< > , relop >
< b , identifier >
< ) , paranthesis >
< printf , keyword >
< ( , paranthesis >
< b , identifier >
< ) , paranthesis >
< ; , seperator >
< printf , keyword >
< ( , paranthesis >
< "how-are-you" , literal >
< ) , paranthesis >
< ; , seperator >
< return , keyword >
< 0 , literal >
< ; , seperator >
< } , paranthesis >

abhishek@hephaestus:~/Desktop/S7/CD LAB$
```

```
Reading from input.c
int main()
{
    int a, b, c;
    c = a + b;
    string s;
    if (a > b)
        printf(b);
    printf("how-are-you");
    return 0;
}
< int , keyword >
< main , identifier >
< ( , paranthesis >
< ) , paranthesis >
< { , paranthesis >
< int , keyword >
< a , identifier >
< , , seperator >
< b , identifier >
< , , seperator >
< c , identifier >
< ; , seperator >
< c , identifier >
< = , operator_b >
< a , identifier >
< + , operator_b >
< b , identifier >
< ; , seperator >
< string , keyword >
< s , identifier >
< ; , seperator >
< if , keyword >
< ( , paranthesis >
< a , identifier >
< > , relop >
< b , identifier >
< ) , paranthesis >
< printf , keyword >
< ( , paranthesis >
< b , identifier >
< ) , paranthesis >
< ; , seperator >
< printf , keyword >
< ( , paranthesis >
< "how-are-you" , literal >
< ) , paranthesis >
< ; , seperator >
< return , keyword >
< 0 , literal >
< ; , seperator >
< } , paranthesis >
```

1.6 Result

Implemented the program to develop a lexical analyzer for C language in CPP. It was compiled using g++ version 9.3.0, and executed in Ubuntu 20.04 and the above output was obtained. The input file is read word by word. The words are further divided using the help of delimiters to form meaningful tokens. After proper pre-processing of read words, we get all the required tokens. All these tokens are tested for keywords, operators(relative operator,binary,unary), literals, delimiters(paranthesis, seperator) and identifiers. They are checked and output is displayed.

Exp 2

2 Lexical analyzer

2.1 Aim

Implement a Lexical analyzer using Lex Tool

2.2 Theory

Lexical analyzer.

Lexical Analysis is the first phase of the compiler also known as a scanner. It converts the High level input program into a sequence of **Tokens**.

- Lexical Analysis can be implemented with the Deterministic finite Automata.
- The output is a sequence of tokens that is sent to the parser for syntax analysis

What is a token?

A lexical token is a sequence of characters that can be treated as a unit in the grammar of the programming languages.

Example of tokens:

1. Type token (id, number, real, . . .)
2. Punctuation tokens (IF, void, return, . . .)
3. Alphabetic tokens (keywords)

Flex (Fast Lexical Analyzer Generator).

FLEX (fast lexical analyzer generator) is a tool/computer program for generating lexical analyzers (scanners or lexers) written by Vern Paxson in C around 1987. It is used together with Berkeley Yacc parser generator or GNU Bison parser generator. Flex and Bison both are more flexible than Lex and Yacc and produces faster code. Bison produces parser from the input file provided by the user. The function *yylex()* is automatically generated by the flex when it is provided with a .l file and this *yylex()* function is expected by parser to call to retrieve tokens from current/this token stream.

Note: The function *yylex()* is the main flex function which runs the Rule Section and extension (.l) is the extension used to save the programs.

2.2.1 Program Structure:

In the input file, there are 3 sections:

1. Definition Section: The definition section contains the declaration of variables, regular definitions, manifest constants. In the definition section, text is enclosed in "% %" brackets. Anything written in this brackets is copied directly to the file lex.yy.c

2. Rules Section: The rules section contains a series of rules in the form: pattern action and pattern must be unintended and action begin on the same line in brackets. The rule section is enclosed in "%% %%".

3. User Code Section: This section contain C statements and additional functions. We can also compile these functions separately and load with the lexical analyzer.

2.3 Algorithm

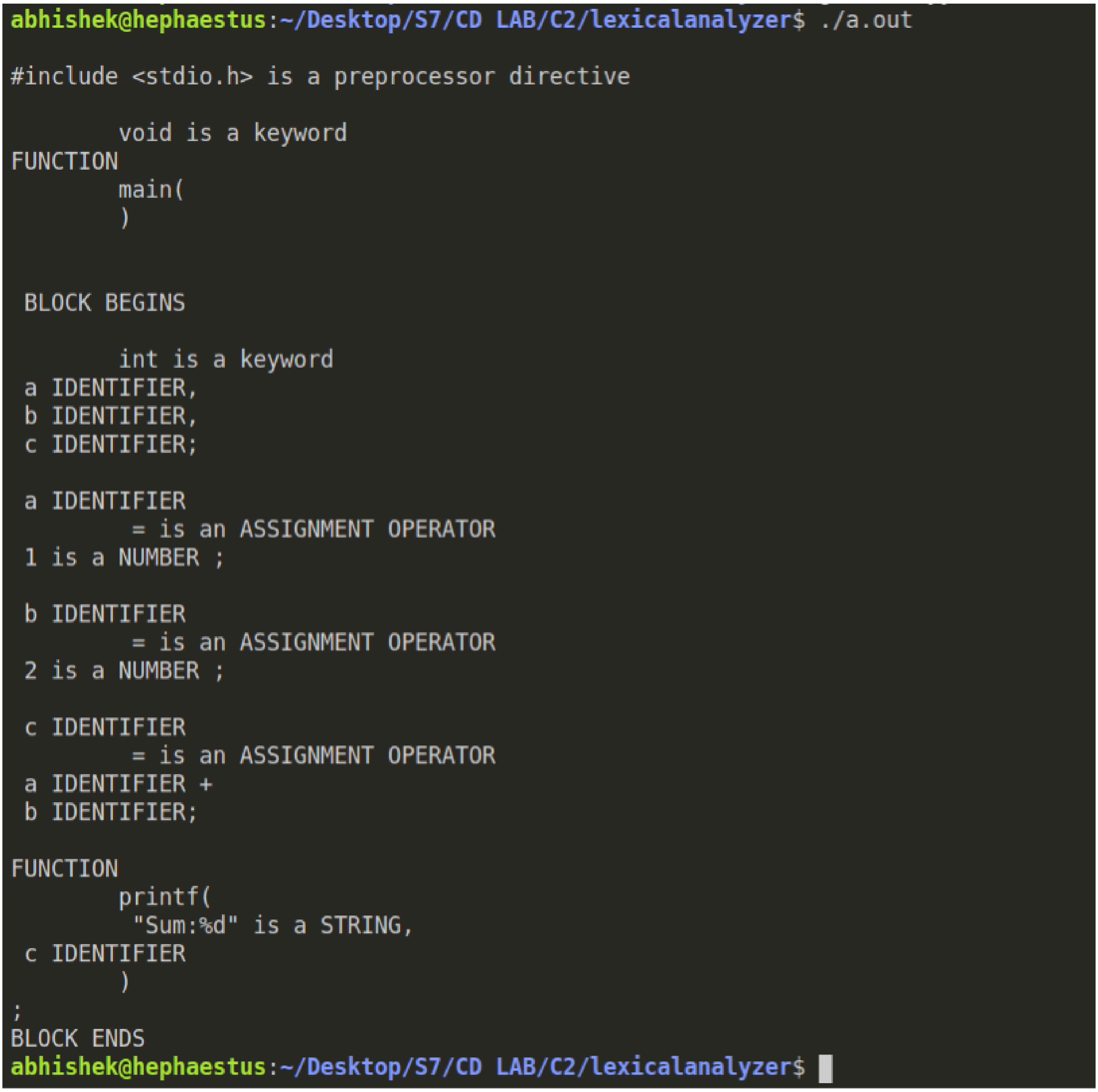
Algorithm 2: Algorithm for Lexical alanyzer

```
Step1: 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,
Step2: 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
Step3: 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.
Step4: Each pattern may have a corresponding action, that is, a fragment of C source code to execute when the pattern is matched.
Step5: 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.
Step6: In user subroutine section, main routine calls yylex(). yywrap() is used to get more input.
Step7: 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.
```

2.4 Code

```
%{
int COMMENT=0;
%}
identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* {printf("\n%s is a preprocessor directive",yytext);}
int |
float |
char |
double |
while |
for |
struct |
typedef |
do |
if |
break |
continue |
void |
switch |
return |
else |
goto {printf("\n\t%s is a keyword",yytext);}
"/*" {COMMENT=1;}{printf("\n\t %s is a COMMENT",yytext);}
{identifier}\( {if(!COMMENT)printf("\nFUNCTION \n\t%s",yytext);}
\{  {if(!COMMENT)printf("\n BLOCK BEGINS");}
\}  {if(!COMMENT)printf("BLOCK ENDS ");}
{identifier}(\[[0-9]*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}
\".*\" {if(!COMMENT)printf("\n\t %s is a STRING",yytext);}
[0-9]+ {if(!COMMENT) printf("\n %s is a NUMBER ",yytext);}
\)(\:)? {if(!COMMENT)printf("\n\t");ECHO;printf("\n");}
\( ECHO;
= {if(!COMMENT)printf("\n\t %s is an ASSIGNMENT OPERATOR",yytext);}
\<= |
\>= |
\< |
== |
\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}
%%
int main(int argc, char **argv)
{
FILE *file;
file=fopen("var.c","r");
if(!file)
{
printf("could not open the file");
exit(0);
}
yyin=file;
yylex();
printf("\n");
return(0);
}
int yywrap()
{
return(1);
}
```

2.5 Output



```
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/lexicalanalyzer$ ./a.out

#include <stdio.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
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/lexicalanalyzer$
```

```
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/lexicalanalyzer$ ./a.out

#include <stdio.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
```

2.6 Result

Implemented the program for Lexical analyzer using lex tool. It was compiled using gcc version 9.3.0,flex 2.6.4 and executed in Ubuntu 20.04 and the above output was obtained.

Exp 3

3 YACC

3.1 Aim

Generate YACC specification for a few syntactic categories.

3.2 Theory

YACC.

A parser generator is a program that takes as input a specification of a syntax, and produces as output a procedure for recognizing that language. Historically, they are also called compiler-compilers.

YACC (yet another compiler-compiler) is an LALR(1) (LookAhead, Left-to-right, Rightmost derivation producer with 1 lookahead token) parser generator. YACC was originally designed for being complemented by Lex.

3.3 Algorithm

Algorithm 3: General Algorithm / Structure of YACC

```
/* definitions */
 ....

%%
/* rules */
....
%%

/* auxiliary routines */
....
```

3.4 Arithmetic Expression

3.4.1 Question

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

3.4.2 Algorithm

Algorithm 4: Algorithm to check valid arithmetic expression.

Step1: Start the program
Step2: Reading an expression
Step3: Checking the validating of the given expression according to the rule using yacc.
Step4: Using expression rule print the result of the given values
Step5: Stop the program

3.4.3 Code

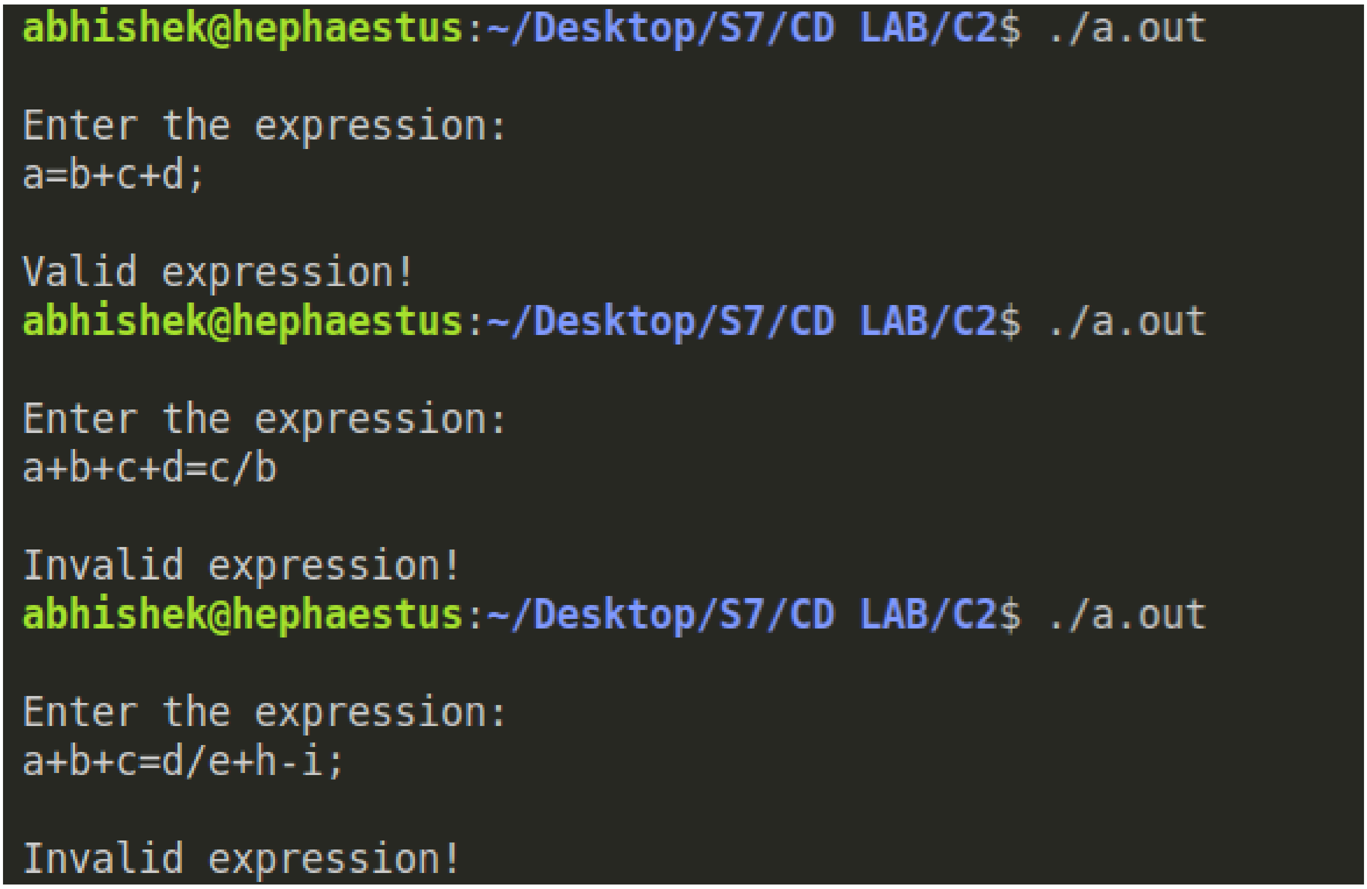
```
%{

    #include<stdio.h>

    int valid=1;

%}

```

```
%token num id op

%%

start : id ’=’ s ’;’

s :     id x

      | num x

      | ’-’ num x

      | ’(’ s ’)’ x

      ;

x :     op s

      | ’-’ s

      |

      ;

%%

int yyerror()

{

    valid=0;

    printf("\nInvalid expression!\n");

    return 0;

}

int main()

{

    printf("\nEnter the expression:\n");

    yyparse();

    if(valid)

    {

        printf("\nValid expression!\n");

    }

}
```

```
%{

    #include "y.tab.h"

%}

%%

[a-zA-Z_][a-zA-Z_0-9]* return id;

[0-9]+(\.[0-9]*)?      return num;

[+/*]                  return op;

.                      return yytext[0];

\n                     return 0;

%%

int yywrap()
```

```

{

return 1;

}
```

3.4.4 Output



```
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2$ ./a.out

Enter the expression:
a=b+c+d;

Valid expression!
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2$ ./a.out

Enter the expression:
a+b+c+d=c/b

Invalid expression!
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2$ ./a.out

Enter the expression:
a+b+c=d/e+h-i;

Invalid expression!
```

```
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2$ ./a.out

Enter the expression:
a=b+c+d;

Valid expression!
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2$ ./a.out

Enter the expression:
a+b+c+d=c/b

Invalid expression!
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2$ ./a.out

Enter the expression:
a+b+c=d/e+h-i;

Invalid expression!
```

3.5 Identifier

3.5.1 Question

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

3.5.2 Algorithm

Algorithm 5: Algorithm to identify identifier

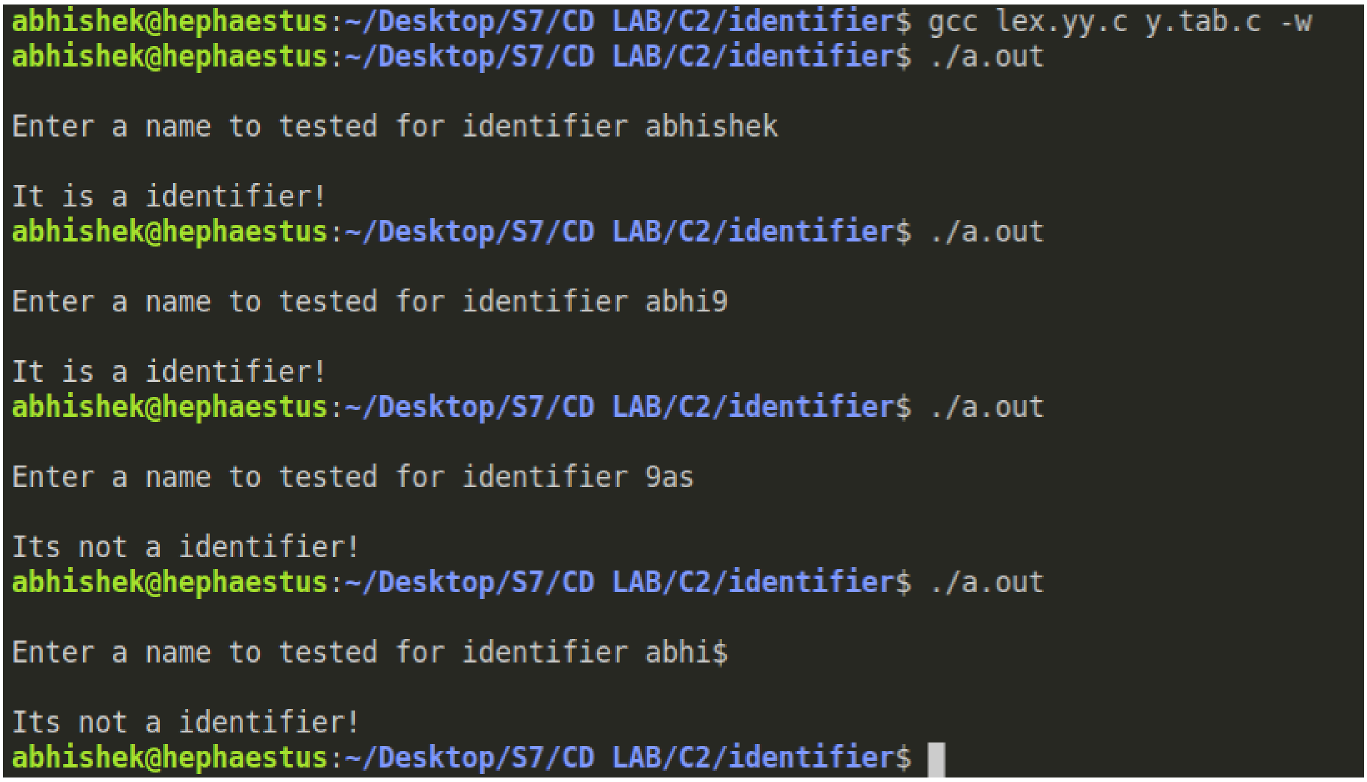
```
Step1: Start the program
Step2: Reading an expression
Step3: Checking the validating of the given expression according to the rule using yacc.
Step4: Using expression rule print the result of the given values
Step5: Stop the program
```

3.5.3 Code

```
%{

    #include<stdio.h>

    int valid=1;

%}

%token digit letter

%%

start : letter s

s :     letter s

      | digit s

      |

      ;

%%

int yyerror()

{

    printf("\nIts not a identifier!\n");

    valid=0;

    return 0;

}

int main()

{

    printf("\nEnter a name to tested for identifier ");

    yyparse();

    if(valid)

    {

        printf("\nIt is a identifier!\n");

    }

}
```

```
%{

    #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;

}
```

3.5.4 Output



```
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ gcc lex.yy.c y.tab.c -w
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ ./a.out

Enter a name to tested for identifier abhishek

It is a identifier!
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ ./a.out

Enter a name to tested for identifier abhi9

It is a identifier!
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ ./a.out

Enter a name to tested for identifier 9as

Its not a identifier!
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ ./a.out

Enter a name to tested for identifier abhi$

Its not a identifier!
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$
```

```
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ lex id.l
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ yacc -d id.y
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ gcc lex.yy.c y.tab.c -w
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ ./a.out

Enter a name to tested for identifier abhishek

It is a identifier!
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ ./a.out

Enter a name to tested for identifier abhi9

It is a identifier!
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ ./a.out
```

```
Enter a name to tested for identifier 9as

Its not a identifier!
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/identifier$ ./a.out

Enter a name to tested for identifier abhi$

Its not a identifier!
```

3.6 Calculator

3.6.1 Question

Implementation of Calculator using LEX and YACC.

3.6.2 Algorithm

Algorithm 6: Algorithm for calculator

Step1: A Yacc source program has three parts as follows:
Declarations

Step2: Declarations Section: This section contains entries that:
i. Include standard I/O header file.
ii. Define global variables.
iii. Define the list rule as the place to start processing.
iv. Define the tokens used by the parser. v. Define the operators and their precedence.

Step3: Rules Section: The rules section defines the rules that parse the input stream. Each rule of a grammar production and the associated semantic action.

Step4: Programs Section: The programs section contains the following subroutines. Because these subroutines are included in this file, it is not necessary to use the yacc library when processing this file.

Step5: Main- The required main program that calls the yyparse subroutine to start the program.

Step6: yyerror(s) -This error-handling subroutine only prints a syntax error message.

Step7: yywrap -The wrap-up subroutine that returns a value of 1 when the end of input occurs. The calc.lex file contains include statements for standard input and output, as programmar file information if we use the -d flag with the yacc command. The y.tab.h file contains definitions for the tokens that the parser program uses.

Step8: calc.lex contains the rules to generate these tokens from the input stream.

3.6.3 Code

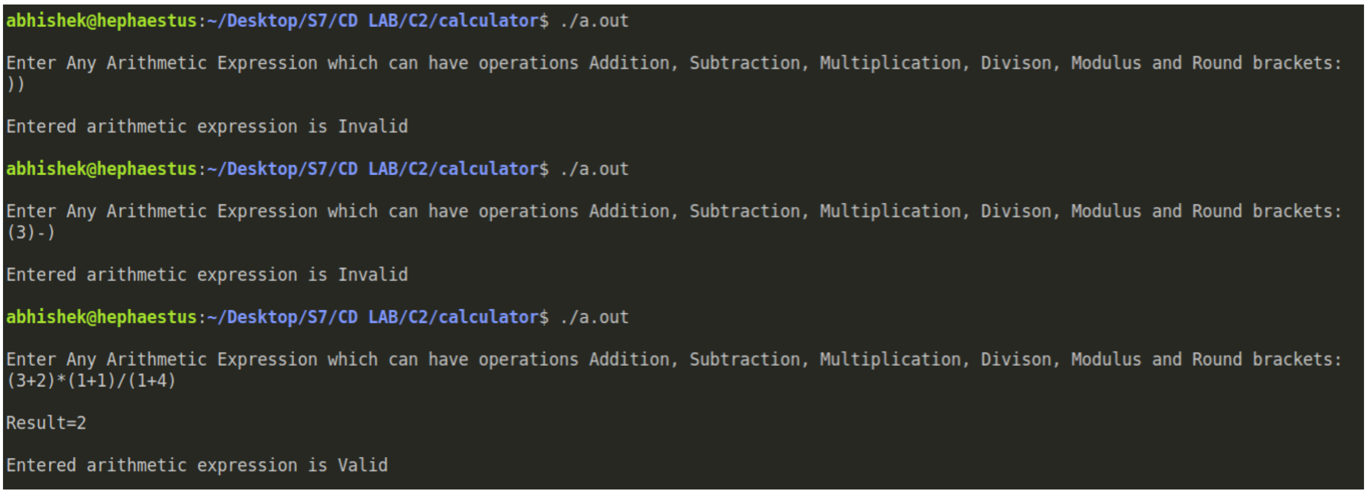
```
%{

    #include<stdio.h>

    int flag=0;



%}

%token NUMBER



%left ’+’ ’-’

%left ’*’ ’/’ ’%’

%left ’(’ ’)’

%%

ArithmeticExpression: E{

         printf("\nResult=%d\n",$$);

         return 0;
```

```

        };

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

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

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

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

 |E'%'E {$$=$1%$3;}

 |'('E')' {$$=$2;}

 | NUMBER {$$=$1;}

;

%%



void main()

{

   printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:\n");

   yyparse();

  if(flag==0)

   printf("\nEntered arithmetic expression is Valid\n\n");



}

void yyerror()

{

   printf("\nEntered arithmetic expression is Invalid\n\n");

   flag=1;

}
```

```
%{

#include<stdio.h>

#include "y.tab.h"

extern int yylval;

%}



%%

[0-9]+ {

          yylval=atoi(yytext);

          return NUMBER;

       }

[\t] ;

[\n] return 0;

. return yytext[0];
```

```

%%

int yywrap()

{

return 1;

}
```

3.6.4 Output



```
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/calculator$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:
))

Entered arithmetic expression is Invalid

abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/calculator$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:
(3)-)

Entered arithmetic expression is Invalid

abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/calculator$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:
(3+2)*(1+1)/(1+4)

Result=2

Entered arithmetic expression is Valid
```

```
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/calculator$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Diviso
))

Entered arithmetic expression is Invalid

abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/calculator$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Diviso
(3)-)

Entered arithmetic expression is Invalid

abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/calculator$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Diviso
(3+2)*(1+1)/(1+4)

Result=2

Entered arithmetic expression is Valid

abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/calculator$
```

3.7 BNF to YACC

3.7.1 Question

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

3.7.2 Algorithm

Algorithm 7: Algorithm for BNF to YACC

```
Step1: Reading an expression.
Step2: Calculate the value of given expression
Step3: Display the value of the nodes based on the precedence.
Step4: Using expression rule print the result of the given values
```

3.7.3 Code

```
%{

#include<string.h>

#include<stdio.h>

struct quad

{

char op[5];

char arg1[10];

char arg2[10];

char result[10];

}QUAD[30];

struct stack

{





int items[100];

int top;

}stk;

int Index=0,tIndex=0,StNo,Ind,tInd;

extern int LineNo;

%}

%union

{

char var[10];

}

%token <var> NUM VAR RELOP

%token MAIN IF ELSE WHILE TYPE

%type <var> EXPR ASSIGNMENT CONDITION IFST ELSEST WHILELOOP

%left '-' '+'

%left '*' '/'
```

```

%%

PROGRAM : MAIN BLOCK

;

BLOCK: ’{’ CODE ’}’

;

CODE: BLOCK

| STATEMENT CODE

| STATEMENT

;

STATEMENT: DESCT ’;’

| ASSIGNMENT ’;’

| 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






;
```

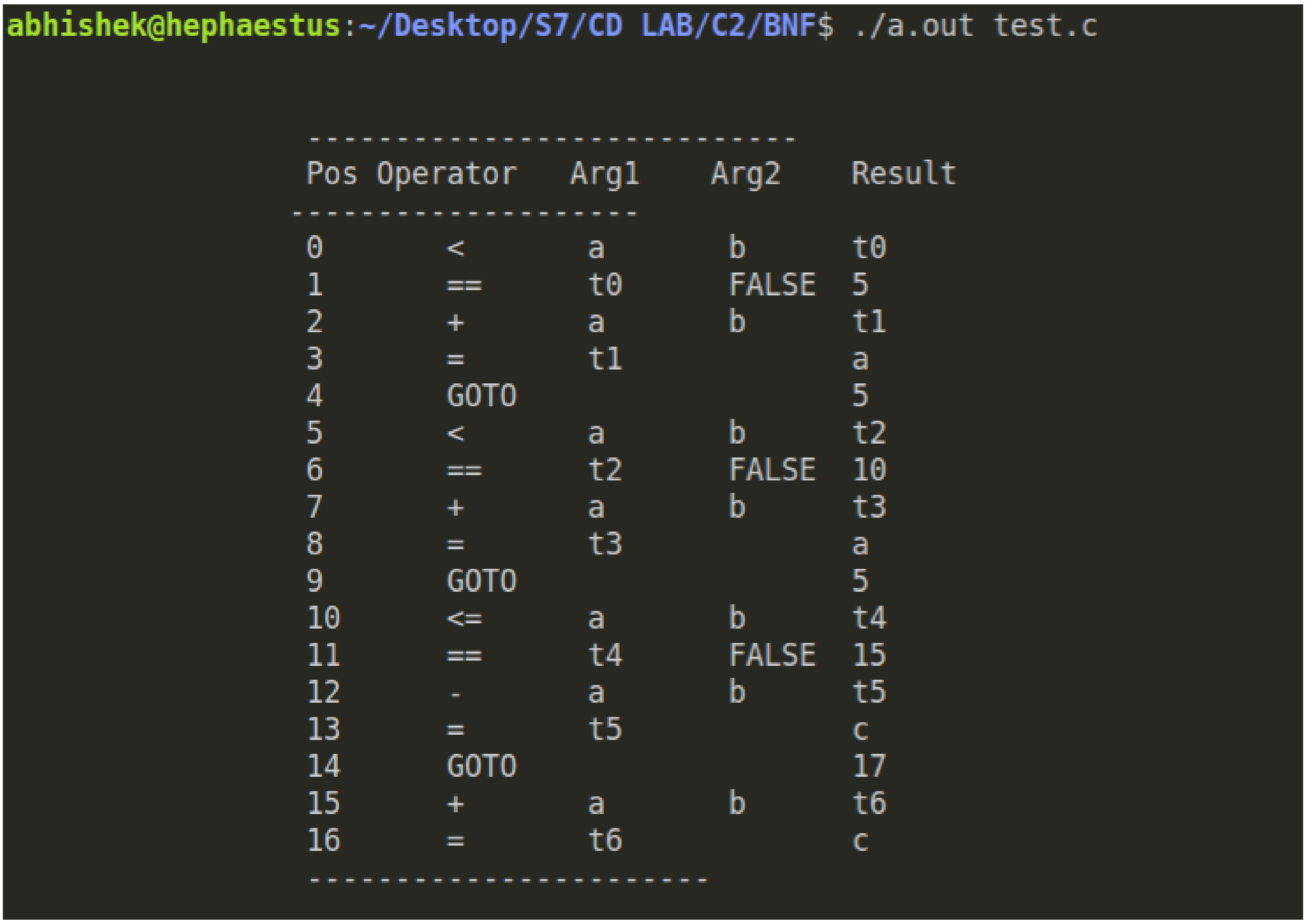
```

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");
```

```c

exit(0);

}

yyin=fp;

}

yyparse();

printf("\n\n\t\t ----------------------------""\n\t\t Pos Operator \tArg1 \tArg2 \tResult" "\n\t\t--------------------");

for(i=0;i<Index;i++)

{

printf("\n\t\t %d\t %s\t %s\t %s\t%s",i,QUAD[i].op,QUAD[i].arg1,QUAD[i].arg2,QUAD[i].result);

}

printf("\n\t\t -----------------------");

printf("\n\n"); return 0; }

void push(int data)

{ stk.top++;

if(stk.top==100)

{

printf("\n Stack overflow\n");

exit(0);

}

stk.items[stk.top]=data;

}

int pop()

{

int data;




if(stk.top==-1)

{

printf("\n Stack underflow\n");

exit(0);

}

data=stk.items[stk.top--];

return data;

}

void AddQuadruple(char op[5],char arg1[10],char arg2[10],char result[10])

{

strcpy(QUAD[Index].op,op);

strcpy(QUAD[Index].arg1,arg1);

strcpy(QUAD[Index].arg2,arg2);

sprintf(QUAD[Index].result,"t%d",tIndex++);

strcpy(result,QUAD[Index++].result);

}

yyerror()

{

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

}
```

```
%{

#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];

%%
```

3.7.4 Output



```
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/BNF$ ./a.out test.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
                -----------------------
```

```
abhishek@hephaestus:~/Desktop/S7/CD LAB/C2/BNF$ ./a.out test.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
                -----------------------
```

3.8 Result

Implemented YACC specification for a few syntactic categories. It was compiled using gcc version 9.3.0,flex 2.6.4,bison (GNU Bison) 3.5.1 and executed in Ubuntu 20.04 and the above output was obtained.

Exp 4

4 ϵ- closure

4.1 Aim

Write program to find ϵ - closure of all states of any given NFA with ϵ transition

4.2 Theory

NFA

- NFA stands for non-deterministic finite automata. It is easy to construct an NFA than DFA for a given regular language.
- The finite automata are called NFA when there exist many paths for specific input from the current state to the next state.
- Every NFA is not DFA, but each NFA can be translated into DFA.
- NFA is defined in the same way as DFA but with the following two exceptions, it contains multiple next states, and it contains ϵ transition.
  - Q: finite set of states
  - Σ: finite set of the input symbol
  - q0: initial state
  - F: final state
  - δ: Transition function
  - δ : Q x $\Sigma \rightarrow 2^Q$

Epsilon Closure:
Epsilon closure for a given state X is a set of states which can be reached from the states X with only (null) or ϵ moves including the state X itself. In other words,ϵ -closure for a state can be obtained by union operation of the ϵ-closure of the states which can be reached from X with a single ϵ move in recursive manner.

4.3 Algorithm

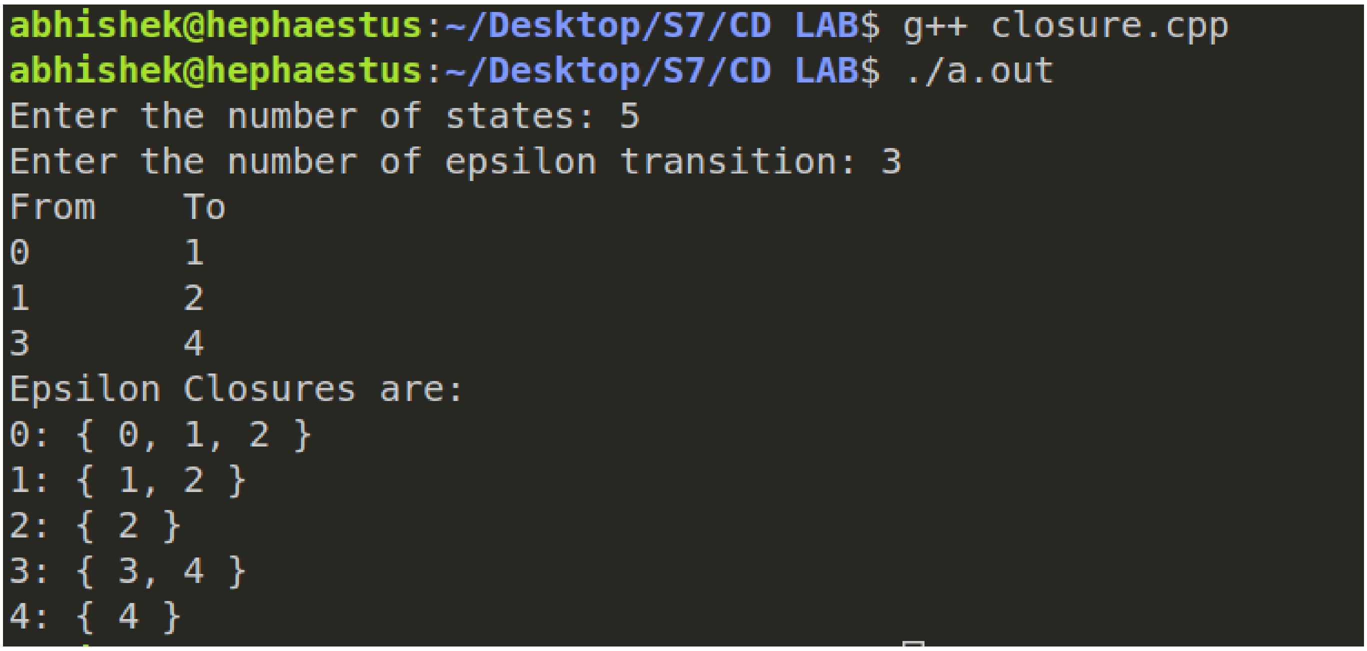
Algorithm 8: Algorithm to find Epsilon Closure

```
function EPSILONCLOSURE ( enfa , int k )
    Initialize a list t containing only state k
    Initialize an iterator to the first element of the list t
    while iterator has not crossed the last element of the list t
        Append all states in the i pair in the transition table of
        enfa which is not previously present in list t to t
        Set the iterator to the next element of the list t
        Return list t as the epsilon - closure for state k in
        epsilon - N F A enfa
end function
```

4.4 Code

```cpp
// CPP program to find the epsilon closure of all states
#include <bits/stdc++.h>
using namespace std;

int main()
{
    cout << "Enter the number of states: ";
    int n, m;
    cin >> n;
    int start, last, temp, temp1;
    int epsilon;
    cout << "Enter the number of epsilon transition: ";
    cin >> epsilon;
    vector<pair<int, int>> eps;
    cout << "From\tTo" << endl;
    for (int i = 0; i < epsilon; ++i)
    {
        cin >> temp >> temp1;
        if (temp >= n || temp < 0 || temp1 >= n || temp1 < 0)
        {
            cout << "incorrect input: program forced to terminate" << endl;
            return 0;
        }
        eps.push_back({temp, temp1});
    }

    queue<int> check;
    cout << "Epsilon Closures are: \n";
    for (int i = 0; i < n; ++i)
    {
        vector<int> visited(n, 0);
        if (visited[i] != 1)
        {
            cout << i << ": ";
            visited[i] = 1;
            cout << "{ " << i;
            for (auto x : eps)
            {
                if (x.first == i)
                {
                    check.push(x.second);
                }
            }
            while (!check.empty())
            {
                int c = check.front();
                check.pop();
                visited[c] = 1;
                cout << ", " << c;
                for (auto x : eps)
                {
                    if (x.first == c && visited[x.second] != 1)
                    {
                        check.push(x.second);
                    }
                }
            }
            cout << " }" << endl;
        }
    }

    return 0;
}
```

4.5 Output



```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ g++ closure.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the number of states: 5
Enter the number of epsilon transition: 3
From    To
0       1
1       2
3       4
Epsilon Closures are:
0: { 0, 1, 2 }
1: { 1, 2 }
2: { 2 }
3: { 3, 4 }
4: { 4 }
```

```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the number of states: 5
Enter the number of epsilon transition: 3
From    To
0       1
1       2
3       4
Epsilon Closures are:
0: { 0, 1, 2 }
1: { 1, 2 }
2: { 2 }
3: { 3, 4 }
4: { 4 }
```

4.6 Result

Implemented the program to find epsilon closure of states of a NFA in CPP. It was compiled using g++ version 9.3.0, and executed in Ubuntu 20.04 and the above output was obtained.

Exp 5

5 ϵ- NFA to NFA

5.1 Aim

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

5.2 Theory

NFA

- NFA stands for non-deterministic finite automata. It is easy to construct an NFA than DFA for a given regular language.
- The finite automata are called NFA when there exist many paths for specific input from the current state to the next state.
- Every NFA is not DFA, but each NFA can be translated into DFA.
- NFA is defined in the same way as DFA but with the following two exceptions, it contains multiple next states, and it contains ϵ transition.
  - Q: finite set of states
  - Σ: finite set of the input symbol
  - q0: initial state
  - F: final state
  - δ: Transition function
  - δ : Q x $\Sigma \rightarrow 2^Q$

5.3 Algorithm

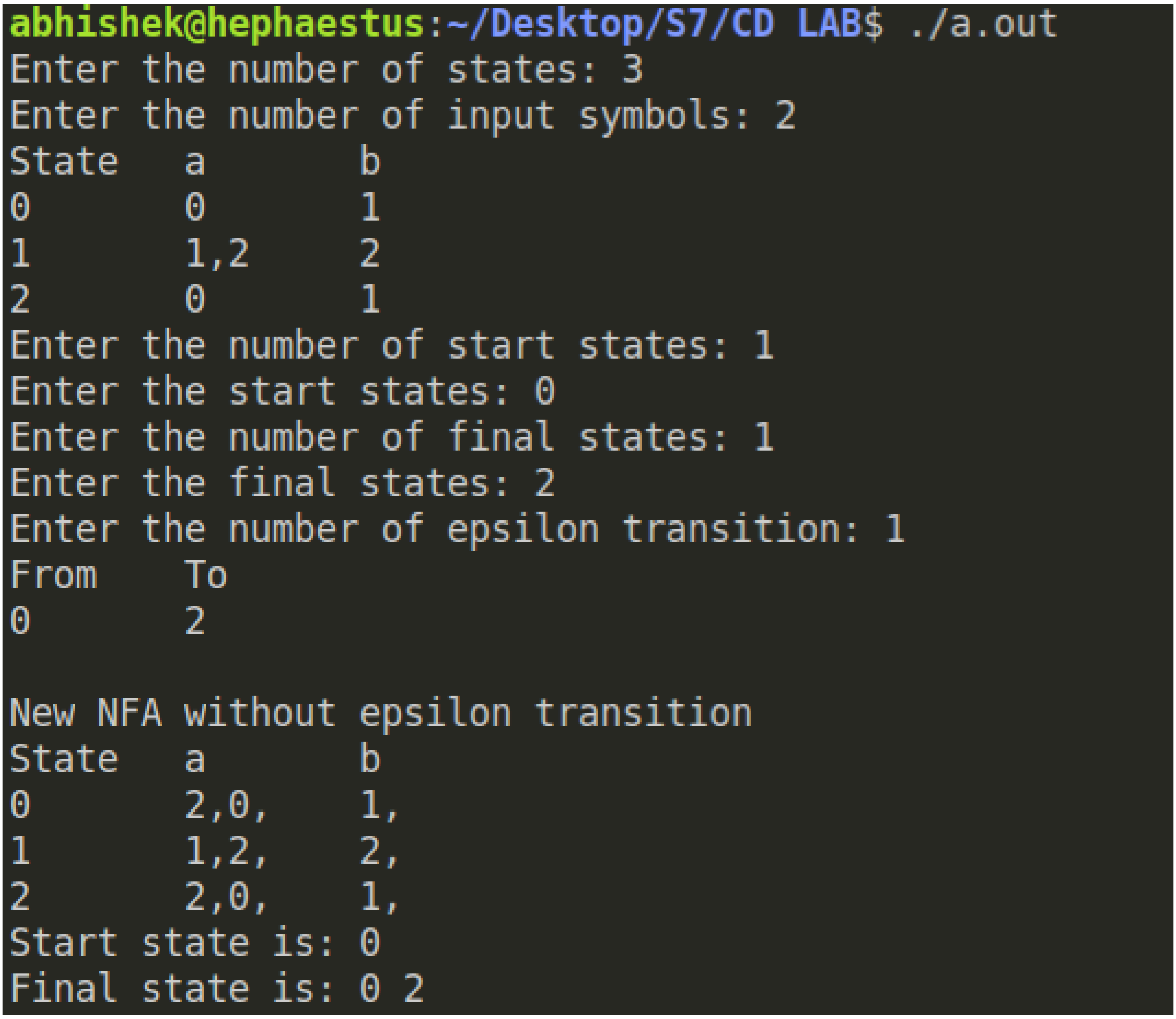
Algorithm 9: Algorithm to convert ϵ NFA to NFA

```
function CONVERTTONFA ( enfa )
    Initialize an empty object of type NFA with variable name t
    Initialize t. numstates = enfa. numstates ,
    t. numalphabets = enfa .numalphabets and
    t. finalstates = enfa . finalstates
    Iterate through each of the state i in Q
        Initialize l to the closure of state i of e - N F A enfa
        Iterate through each of the input symbol j in Z
            Initialize an empty list of states f
            Iterate through each state k in l
            Add all states of enfa . transitiontable [ k ][ j + 1] to f
        Remove all the duplicates from f
        Compute the e - closure c of f
        Set t . transitiontable [ i ][ j ] = c
    Return t as the NFA without e - transitions corresponding to the
    e - NFA enfa
end function
```

5.4 Code

```cpp
// CPP program to covert epsilon nfa to non epsilon nfa
#include <bits/stdc++.h>
using namespace std;


void print_header(int m)
{
    cout << "State";
    for (int j = 0; j < m; ++j)
    {
        char ch = char(int(’a’) + j);
        cout << "\t" << ch;
    }
    cout << endl;
}

unordered_set<int> split_Add(string s)
{
    int size = s.size();
    unordered_set<int> res;
    int start = 0;
    for (int i = 0; i < size; ++i)
    {
        if (s[i] == ’,’)
        {
            string sub = s.substr(start, i - start);
            start = i + 1;
            int temp = stoi(sub);
            res.insert(temp);
        }
    }
    string sub = s.substr(start, size - start);
    int temp = stoi(sub);
    res.insert(temp);
    return res;
}
int main()
{
    cout << "Enter the number of states: ";
    int n, m;
    cin >> n;
    cout << "Enter the number of input symbols: ";
    cin >> m;
    vector<vector<unordered_set<int>>> table(n, vector<unordered_set<int>>(m));
    print_header(m);
    for (int i = 0; i < n; ++i)
    {
        cout << i << "\t";
        for (int j = 0; j < m; ++j)
        {
            string s;
            cin >> s;
            unordered_set<int> to_states;
            if (s != "-")
                to_states = split_Add(s);
            table[i][j] = to_states;
        }
    }

    int start, last, temp, temp1;
    unordered_set<int> s, f;
    cout << "Enter the number of start states: ";
    cin >> start;
    cout << "Enter the start states: ";
    for (int i = 0; i < start; ++i)
    {
        cin >> temp;
        s.insert(temp);
    }
    cout << "Enter the number of final states: ";
    cin >> last;
    cout << "Enter the final states: ";
    for (int i = 0; i < last; ++i)
    {
        cin >> temp;
        f.insert(temp);
    }
    int epsilon;
    cout << "Enter the number of epsilon transition: ";
    cin >> epsilon;
    vector<pair<int, int>> eps;
    cout << "From\tTo" << endl;
    for (int i = 0; i < epsilon; ++i)
    {
        cin >> temp >> temp1;
        eps.push_back({temp, temp1});
    }

    queue<int> check;
    vector<unordered_set<int>> closures; // to store the set of states
    // cout << "Epsilon Closures are: \n";
    for (int i = 0; i < n; ++i)
    {
        vector<int> visited(n, 0);
        if (visited[i] != 1)
        {
            unordered_set<int> building;
            building.insert(i);
            //cout << i << ": ";
            visited[i] = 1;
            //cout << "{ " << i;
            for (auto x : eps)
            {
                if (x.first == i)
                {
                    check.push(x.second);
                }
            }
            while (!check.empty())
            {
                int c = check.front();
                check.pop();
                building.insert(c);
                visited[c] = 1;
                //cout << ", " << c;
                for (auto x : eps)
                {
                    if (x.first == c && visited[x.second] != 1)
                    {
                        check.push(x.second);
                    }
                }
            }
            closures.push_back(building);
        }
    }
    cout << endl;
    cout << "New NFA without epsilon transition\n";
    print_header(m);
    for (int i = 0; i < n; ++i)
    {
        cout << i << "\t";
        for (int j = 0; j < m; ++j)
        { // for the new state k where each transitions go
            unordered_set<int> check;
            for (auto x : closures[i])
            {
                for (auto y : table[x][j])
                {
                    if (check.find(y) == check.end())
                    {
                        check.insert(y);
                        for (auto z : closures[y])
                        {
                            if (check.find(z) == check.end())
                                check.insert(z);
                        }
                    }
                }
            }
            if (check.empty())
            {
                cout << "-";
            }
            else
            {
                for (auto x : check)
                {
                    cout << x << ",";
                }
            }

            cout << "\t";
        }
        cout << endl;
    }
    cout << "Start state is: ";
    for (int i = 0; i < n; ++i)
    {
        for (auto x : closures[i])
        {
            if (s.find(x) != f.end())
            {
                cout << i << " ";
                break;
            }
        }
    }
    cout << endl;
    cout << "Final state is: ";
    for (int i = 0; i < n; ++i)
    {
        for (auto x : closures[i])
        {
            if (f.find(x) != f.end())
            {
                cout << i << " ";
                break;
            }
        }
    }
    cout << endl;
    return 0;
}
```

5.5 Output



```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the number of states: 3
Enter the number of input symbols: 2
State   a       b
0       0       1
1       1,2     2
2       0       1
Enter the number of start states: 1
Enter the start states: 0
Enter the number of final states: 1
Enter the final states: 2
Enter the number of epsilon transition: 1
From    To
0       2

New NFA without epsilon transition
State   a       b
0       2,0,    1,
1       1,2,    2,
2       2,0,    1,
Start state is: 0
Final state is: 0 2
```

```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the number of states: 3
Enter the number of input symbols: 2
State   a       b
0       0       1
1       1,2     2
2       0       1
Enter the number of start states: 1
Enter the start states: 0
Enter the number of final states: 1
Enter the final states: 2
Enter the number of epsilon transition: 1
From    To
0       2

New NFA without epsilon transition
State   a       b
0       2,0,    1,
1       1,2,    2,
2       2,0,    1,
Start state is: 0
Final state is: 0 2
```

5.6 Result

Implemented the program toconvert ϵ-NFA to a NFA in CPP. It was compiled using g++ version 9.3.0, and executed in Ubuntu 20.04 and the above output was obtained.

Exp 6

6 NFA to DFA

6.1 Aim

Write program to convert NFA to DFA.

6.2 Theory

NFA

- NFA stands for non-deterministic finite automata. It is easy to construct an NFA than DFA for a given regular language.
- The finite automata are called NFA when there exist many paths for specific input from the current state to the next state.
- Every NFA is not DFA, but each NFA can be translated into DFA.
- NFA is defined in the same way as DFA but with the following two exceptions, it contains multiple next states, and it contains ϵ transition.
  - Q: finite set of states
  - Σ: finite set of the input symbol
  - q0: initial state
  - F: final state
  - δ: Transition function
  - δ : Q x $\Sigma \rightarrow 2^Q$

DFA

- In a DFA, for a particular input character, the machine goes to one state only.
- A transition function is defined on every state for every input symbol.
- Also in DFA null (or ϵ) move is not allowed,
- i.e., DFA cannot change state without any input character.
- DFA consists of 5 tuples Q, Σ , q, F, δ.
  - Q : set of all states.
  - Σ : set of input symbols. (Symbols which machine takes as input)
  - q : Initial state. (Starting state of a machine)
  - F : set of final state.
  - δ : Transition Function, defined as δ : Q X $\Sigma \rightarrow$ Q.

6.3 Algorithm

Algorithm 10: Algorithm to convert NFA to DFA

```
function CONVERTTODFA ( nfa )
    Initialize an empty object of type DFA with variable name dfa
    Initialize dfa . num_alphabets = nfa . num _alphabets , i = 0
    Intialize a set lazySet which stores subsets of Q and store {0} in it Create a new row of
    size dfa . num_alphabets and insert into dfa .
    table and initialize all values to -1.
    While i < lazySet . size ()
        Iterate through each of the input symbol j in E
            Initialize an empty set of states reachable and a variable next = -1
            Iterate i through each element in lazySet [ i ] and push
            into reachable the set nfa . table [ i ][ j ]
            Check if reachable is already in lazySet . If yes ,
            the get the value of next from      lazySet . If not , then
                Insert into lazySet , the set reachable and
                set next = lazySet . size ()
                Insert next into dfa . finalStates if any element
                in reachable is a final state of the original nfa
                Create a new row of size dfa . num_alphabets and
                insert into dfa . table and
                initialize all values to -1.
                dfa . table [ i ][ j ] = next
        Increment i
    Return dfa as the DFA .
end function
```

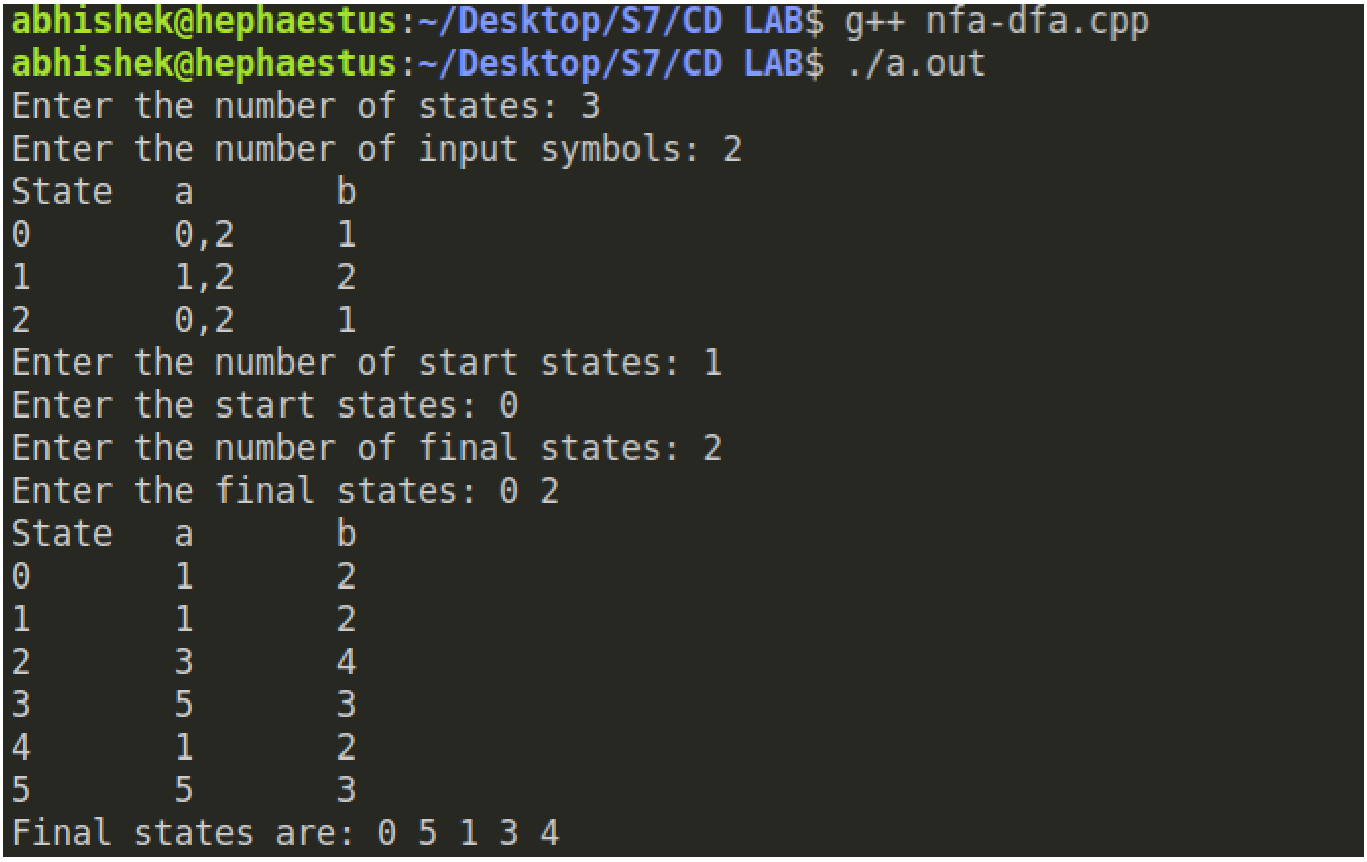
6.4 Code

```
//CPP program to convert NFA into DFA
#include <bits/stdc++.h>
using namespace std;
void print_header(int m)
{
    cout << "State";
    for (int j = 0; j < m; ++j)
    {
        char ch = char(int(’a’) + j);
        cout << "\t" << ch;
    }
    cout << endl;
}
set<int> split_Add(string s)
{
    int size = s.size();
    set<int> res;
    int start = 0;
    for (int i = 0; i < size; ++i)
    {
        if (s[i] == ’,’)
        {
            string sub = s.substr(start, i - start);
            start = i + 1;
            int temp = stoi(sub);
            res.insert(temp);
        }
    }
    string sub = s.substr(start, size - start);
    int temp = stoi(sub);
    res.insert(temp);
    return res;
}
void print(vector<vector<set<int>>> table)
{
    for (auto x : table)
    {
        for (auto y : x)
        {
            for (auto z : y)
            {
                cout << z << ",";
            }
            cout << "\t";
        }
        cout << endl;
    }
}
int main()
{
    cout << "Enter the number of states: ";
    int n, m;
    cin >> n;
    cout << "Enter the number of input symbols: ";
    cin >> m;
    vector<vector<set<int>>> table(n, vector<set<int>>(m));
    print_header(m);
    for (int i = 0; i < n; ++i)
    {
        cout << i << "\t";
        for (int j = 0; j < m; ++j)
        {
            string s;
            cin >> s;
            set<int> to_states;
            if (s != "-")
                to_states = split_Add(s);
            table[i][j] = to_states;
        }
    }
    //print(table);
    int start, last, temp, temp1;
    set<int> s, f;
    cout << "Enter the number of start states: ";
```

```cpp
    cin >> start;
    cout << "Enter the start states: ";
    for (int i = 0; i < start; ++i)
    {
        cin >> temp;
        s.insert(temp);
    }
    cout << "Enter the number of final states: ";
    cin >> last;
    cout << "Enter the final states: ";
    for (int i = 0; i < last; ++i)
    {
        cin >> temp;
        f.insert(temp);
    }
    vector<vector<set<int>>> dfa;
    set<set<int>>
        present;
    map<set<int>, int> state_num;
    queue<set<int>> elem;

    int dfa_state = 0;
    elem.push(s);
    present.insert(s);
    while (!elem.empty())
    {

        vector<set<int>> dfa_row;

        set<int> current = elem.front();
        state_num[current] = dfa_state;
        dfa_state++;
        dfa_row.push_back(current);
        elem.pop();
        for (int i = 0; i < m; ++i)
        {
            set<int> new_state;
            for (auto x : current)
            {
                for (auto y : table[x][i])
                {
                    new_state.insert(y);
                }
            }
            dfa_row.push_back(new_state);
            if (present.find(new_state) == present.end())
            {
                present.insert(new_state);
                elem.push(new_state);
            }
        }
        dfa.push_back(dfa_row);
    }
    print_header(m);
    for (int i = 0; i < dfa_state; ++i)
    {
        for (int j = 0; j < m + 1; ++j)
        {
            cout << state_num[dfa[i][j]] << "\t";
        }
        cout << endl;
    }
    cout << "Final states are: ";
    for (auto x : present)
    {
        for (auto y : x)
        {
            if (f.find(y) != f.end())
            {
                cout << state_num[x] << " ";
                break;
            }
        }
    }
    cout << endl;
    return 0;
```

```
}
```

6.5 Output



```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ g++ nfa-dfa.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the number of states: 3
Enter the number of input symbols: 2
State   a       b
0       0,2     1
1       1,2     2
2       0,2     1
Enter the number of start states: 1
Enter the start states: 0
Enter the number of final states: 2
Enter the final states: 0 2
State   a       b
0       1       2
1       1       2
2       3       4
3       5       3
4       1       2
5       5       3
Final states are: 0 5 1 3 4
```

```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ g++ nfa-dfa.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the number of states: 3
Enter the number of input symbols: 2
State   a       b
0       0,2     1
1       1,2     2
2       0,2     1
Enter the number of start states: 1
Enter the start states: 0
Enter the number of final states: 2
Enter the final states: 0 2
State   a       b
0       1       2
1       1       2
2       3       4
3       5       3
4       1       2
5       5       3
Final states are: 0 5 1 3 4
```

6.6 Result

Implemented the program to convert DFA to a NFA in CPP. It was compiled using g++ version 9.3.0, and executed in Ubuntu 20.04 and the above output was obtained.

Exp 7

7 DFA Minimisation

7.1 Aim

Write program to minimize any given DFA.

7.2 Theory

DFA

- In a DFA, for a particular input character, the machine goes to one state only.
- A transition function is defined on every state for every input symbol.
- Also in DFA null (or ϵ) move is not allowed,
- i.e., DFA cannot change state without any input character.
- DFA consists of 5 tuples Q, Σ , q, F, δ.
  - Q : set of all states.
  - Σ : set of input symbols. (Symbols which machine takes as input)
  - q : Initial state. (Starting state of a machine)
  - F : set of final state.
  - δ : Transition Function, defined as δ : Q X $\Sigma \rightarrow$ Q.

Minimization of DFA

Suppose there is a DFA D $< Q, \Sigma, q0, \delta, F >$which recognizes a language L.
Then the minimized DFA D $< Q^{'}, \Sigma, q0, \delta^{'}, F^{'} >$ can be constructed for language L as:

1. We will divide Q (set of states) into two sets. One set will contain all final states and other set will contain non-final states. This partition is called P0.
2. Initialize k = 1
3. Find Pk by partitioning the different sets of Pk-1. In each set of Pk-1, we will take all possible pair of states. If two states of a set are distinguishable, we will split the sets into different sets in Pk.
4. Stop when Pk = Pk-1 (No change in partition)
5. All states of one set are merged into one. No. of states in minimized DFA will be equal to no. of sets in Pk.

7.3 Algorithm

Algorithm 11: Algorithm to minimize states in a DFA

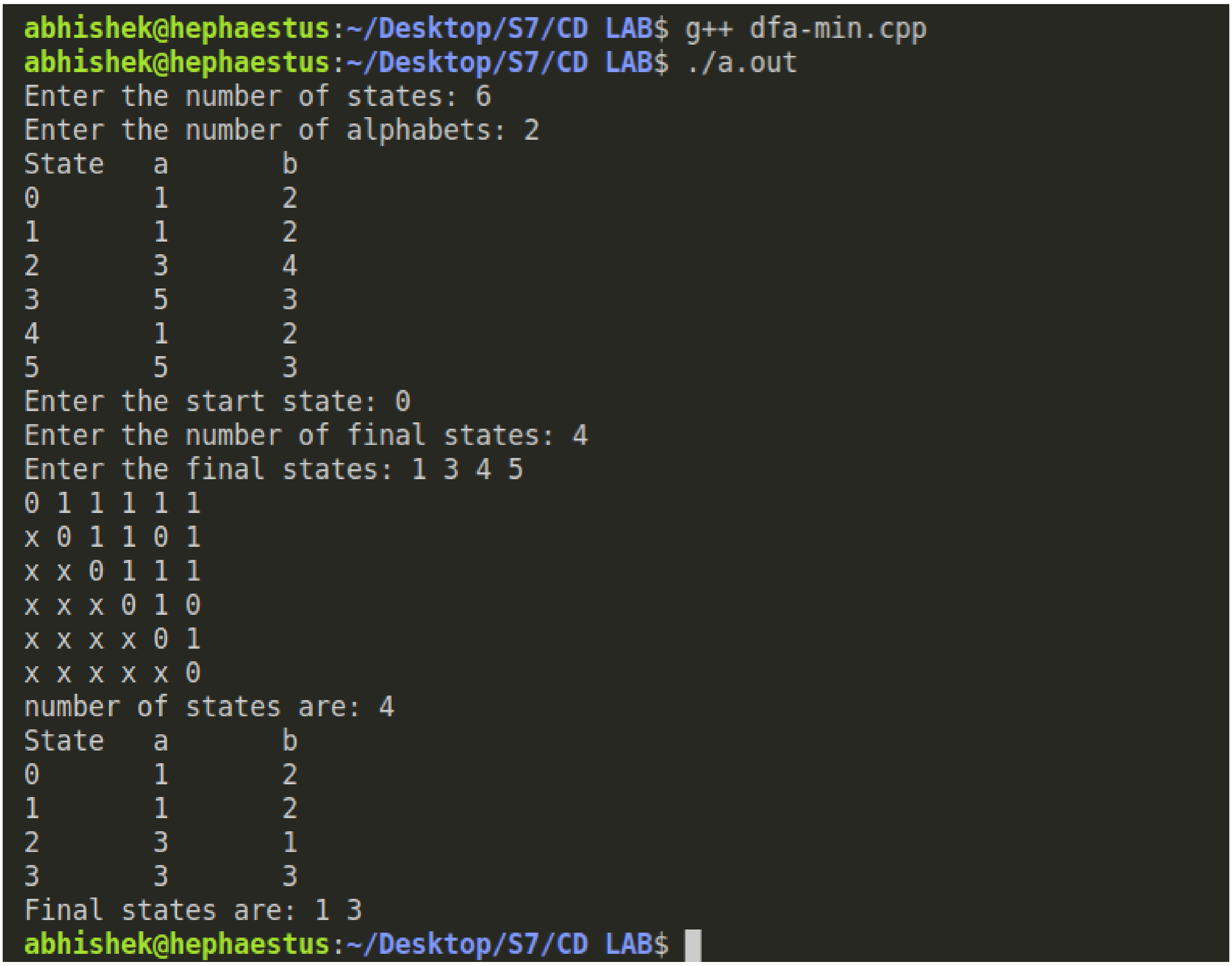
```
function MINIMIZEDFA ( dfa )
    Initialize an empty object of type dfa with variable name minDfa
    Initialize minDfa . num_alphabets = dfa . num_alphabets
    Initialize a matrix m of size a . num_states x a . num_states and
    set every cell in the matrix to 0
    Initialize a flag variable f to 1
    For all state pairs (x , y ) , set m [ x ][ y ] = 1 if x is a final
    state and y is a non - final state or vice - versa ( Choose either
    upper or lower triangle of the matrix ) .
    While f ! = 0
        Set f to 0
        For all states i from 0 to dfa . num_states
        For all states j from i + 1 to dfa . num_states
            If for any symbol u in Z , m [ i ][ j ] = 0 and m [ dfa .
            transitiontable [ i ][ u ]][ dfa . transitiontable [ j ][ u ]] = 1 ,
            Then Set m[i][j] = 1 and f = 1
        Represent those pair of states (a , b ) which has m [ a ][ b ] = 0 by
        a single state a in the minimized DFA minDfa .
    Return minDfa as the minimised DFA .
end function
```

7.4 Code

```cpp
#include <bits/stdc++.h>
using namespace std;
void print_header(int m)
{
    cout << "State";
    for (int j = 0; j < m; ++j)
    {
        char ch = char(int(’a’) + j);
        cout << "\t" << ch;
    }
    cout << endl;
}

vector<vector<int>> input_dfa(int *row, int *column)
{
    int n, m;
    cout << "Enter the number of states: ";
    cin >> n;
    cout << "Enter the number of alphabets: ";
    cin >> m;
    vector<vector<int>> dfa(n, vector<int>(m, 0));
    print_header(m);
    for (int i = 0; i < n; ++i)
    {
        cout << i << "\t";
        for (int j = 0; j < m; ++j)
        {
            cin >> dfa[i][j];
        }
    }
    *row = n;
    *column = m;

    return dfa;
}
int main()
{
    int n, m;
    vector<vector<int>> dfa = input_dfa(&n, &m);
    vector<vector<int>> matrix(n, vector<int>(m, 0));
    int s, last, temp;
    cout << "Enter the start state: ";
    cin >> s;
    set<int> f;
    cout << "Enter the number of final states: ";
    cin >> last;
    cout << "Enter the final states: ";
    for (int i = 0; i < last; ++i)
    {
        cin >> temp;
        f.insert(temp);
    }
    //differentiate final and non finale
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            if (f.find(i) != f.end() && f.find(j) == f.end())
            {
                matrix[i][j] = 1;
            }
            if (f.find(i) == f.end() && f.find(j) != f.end())
            {
                matrix[i][j] = 1;
            }
        }
    }
    int flag = 1;
    while (flag)
    {
        flag = 0;
        for (int i = 0; i < n; ++i)
        {
            for (int j = 0; j < n; ++j)
            {
                for (int k = 0; k < m; ++k)
                {
                    if (matrix[i][j] != 1)
                        if (matrix[dfa[i][k]][dfa[j][k]] == 1)
                        {
                            matrix[i][j] = 1;
                            flag = 1;
                            break;
                        }
                }
            }
        }
    }
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            if (i > j)
            {
                cout << "x"
                     << " ";
            }
            else
            {
                cout << matrix[i][j] << " ";
            }
        }
        cout << endl;
    }
    int num_state = 0;
    vector<int> visited(n, 0);
    vector<set<int>> minimised;
    unordered_map<int, int> mapping;
    for (int i = 0; i < n; ++i)
    {
        set<int> new_state;
        if (visited[i] != 1)
        {
            new_state.insert(i);
            mapping[i] = num_state;
            for (int j = i + 1; j < n; ++j)
            {
                if (matrix[i][j] == 0)
                {
                    new_state.insert(j);
                    mapping[j] = num_state;
                    visited[j] = 1;
                }
```

```cpp
            }
            minimised.push_back(new_state);
            num_state++;
        }
    }
    cout << "number of states are: " << num_state << endl;

    print_header(m);
    for (auto x : minimised)
    {
        for (auto y : x)
        {
            cout << mapping[y] << "\t";
            for (int j = 0; j < m; ++j)
            {
                cout << mapping[dfa[y][j]] << "\t";
            }
            break;
        }
        cout << endl;
    }
    cout << "Final states are: ";
    for (auto x : minimised)
    {
        for (auto y : x)
        {
            if (f.find(y) != f.end())
            {
                cout << mapping[y] << " ";
                break;
            }
        }
    }
    cout << endl;
    return 0;
}
```

7.5 Output



```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ g++ dfa-min.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the number of states: 6
Enter the number of alphabets: 2
State   a       b
0       1       2
1       1       2
2       3       4
3       5       3
4       1       2
5       5       3
Enter the start state: 0
Enter the number of final states: 4
Enter the final states: 1 3 4 5
0 1 1 1 1 1
x 0 1 1 0 1
x x 0 1 1 1
x x x 0 1 0
x x x x 0 1
x x x x x 0
number of states are: 4
State   a       b
0       1       2
1       1       2
2       3       1
3       3       3
Final states are: 1 3
abhishek@hephaestus:~/Desktop/S7/CD LAB$
```

```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ g++ dfa-min.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the number of states: 6
Enter the number of alphabets: 2
State   a       b
0       1       2
1       1       2
2       3       4
3       5       3
4       1       2
5       5       3
Enter the start state: 0
Enter the number of final states: 4
Enter the final states: 1 3 4 5
0 1 1 1 1 1
x 0 1 1 0 1
x x 0 1 1 1
x x x 0 1 0
x x x x 0 1
x x x x x 0
number of states are: 4
State   a       b
0       1       2
1       1       2
2       3       1
```

```
3       3       3
Final states are: 1 3
```

7.6 Result

Implemented the program to minimise a DFA in CPP. It was compiled using g++ version 9.3.0, and executed in Ubuntu 20.04 and the above output was obtained.

Exp 8

8 Operator Precedence Parsing

8.1 Aim

Develop an operator precedence parser for a given language

8.2 Theory

Operator Precedence Parser An operator precedence parser is a bottom-up parser that interprets an operator grammar. This parser is only used for operator grammars. Ambiguous grammars are not allowed in any parser except operator precedence parser. There are two methods for determining what precedence relations should hold between a pair of terminals:

1. Use the conventional associativity and precedence of operator.
2. The second method of selecting operator-precedence relations is first to construct an unambiguous grammar for the language, a grammar that reflects the correct associativity and precedence in its parse trees.

- This parser relies on the following three precedence relations: $\lessdot, \doteq, \gtrdot$
- a $\lessdot$ b This means a “yields precedence to” b.
- a $\gtrdot$ b This means a “takes precedence over” b.
- a $\doteq$ b This means a “has same precedence as” b.

8.3 Algorithm

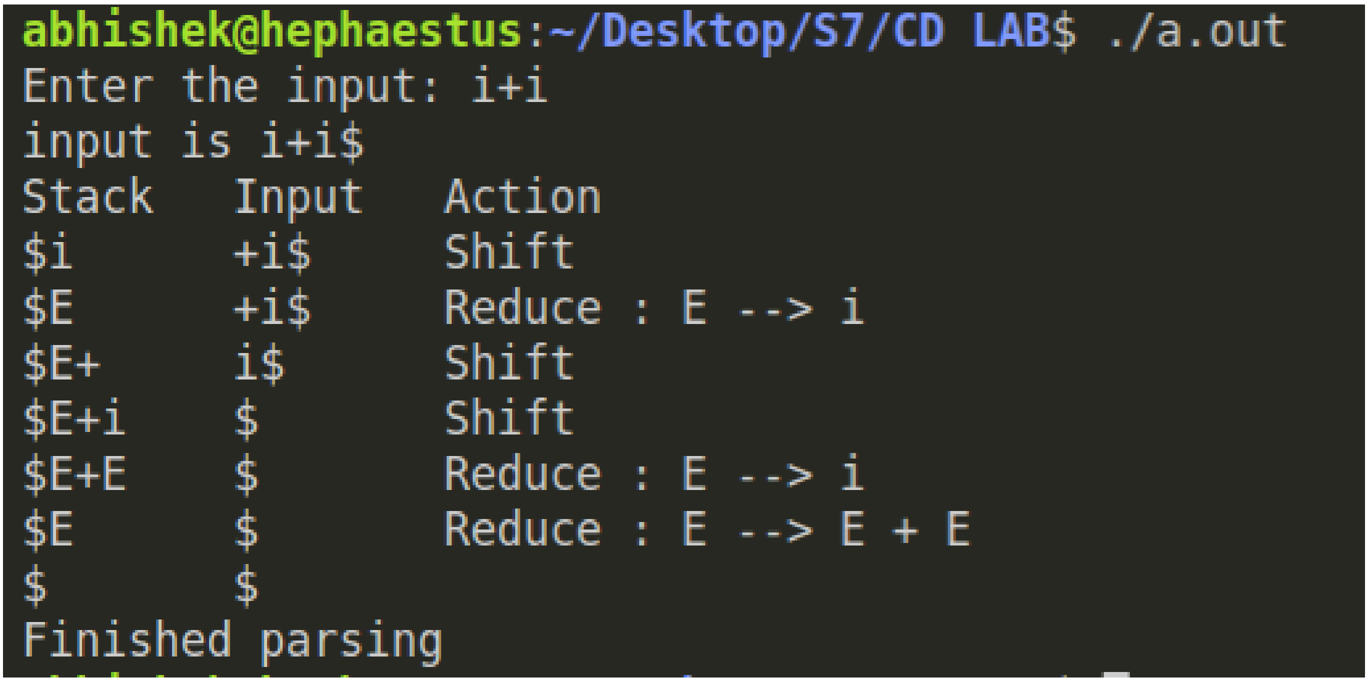
Algorithm 12: Algorithm for precedence parsing

```
if ( a is $ and b is $ )
    return
else
    if a . > b or a =. b then
        push a onto the stack
        advance ip to the next input symbol
    else if a <. b then
        repeat
        c <- pop the stack
        until ( c . > stack - top )
    else error
end
```

8.4 Code

```
#include <bits/stdc++.h>
using namespace std;
void set_precedence(unordered_map<char, int> &precedence)
{
    precedence[’$’] = 0;
    precedence[’(’] = 0;
    precedence[’E’] = 1;
    precedence[’+’] = 3;
    precedence[’*’] = 4;
    precedence[’)’] = 5;
    precedence[’i’] = 5;
}
void print_stack(stack<char> check)
{
    string s = "";
    while (!check.empty())
    {
        s = check.top() + s;
        check.pop();
    }
    cout << s;
}
void print_string(string s, int n)
{
    int size = s.size();
    for (int i = n; i < size; ++i)
    {
        cout << s[i];
    }
}
int main()
{
    unordered_map<char, int> precedence;
    set_precedence(precedence);
    cout << "Enter the input: ";
    string s;
    stack<char> check;
    int ip = 0;
    check.push(’$’);
    cin >> s;
    s += "$";
    cout << "input is " << s << endl;
    cout << "Stack\tInput\tAction" << endl;
    while (true)
    {
        //cout << "in while loop" << endl;
        string action;
        if (s[ip] == ’$’ && check.top() == ’$’)
        {
            cout << "Finished parsing" << endl;
            break;
        }
        if (check.empty() || ip >= s.size())
        {
            cout << "Parsing Completed" << endl;
            break;
        }
        if (s[ip] == ’(’ || precedence[s[ip]] >= precedence[check.top()]) //Push into stack
        {
            //cout << "inside the shifft part" << endl;
            check.push(s[ip]);
            ip++;
            action = "Shift";
        }
        else
        {
            string temp = "";
            while (precedence[s[ip]] < precedence[check.top()])
            {
                char top = check.top();
                temp = top + temp;
                check.pop();
                if (top == ’i’)
                {
                    break;
                }
            }
            if (temp == "i")
            {
                action = "Reduce : E --> i";
                check.push(’E’);
            }
            else if (temp == "E+E")
            {
                action = "Reduce : E --> E + E ";
                check.push(’E’);
            }
            else if (temp == "E*E")
            {
                action = "Reduce : E --> E * E ";
                check.push('E');
            }
            else if (temp == "(E)")
            {
                action = "Reduce : E --> ( E ) ";
                check.push('E');
            }
            else if (temp == "E")
            {
                //nothing
            }
            else
            {
                cout << "unexpected  condition " << temp << endl;
            }
        }
        //cout << "endl of loop" << endl;
        print_stack(check);
        //cout << "stack printing finished " << endl;
        cout << "\t";
        print_string(s, ip);
        cout << "\t";
        cout << action << endl;
    }

    return 0;
}
```

8.5 Output



```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the input: i+i
input is i+i$
Stack   Input   Action
$i      +i$     Shift
$E      +i$     Reduce : E --> i
$E+     i$      Shift
$E+i    $       Shift
$E+E    $       Reduce : E --> i
$E      $       Reduce : E --> E + E
$       $
Finished parsing
```

```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the input: i+i
input is i+i$
Stack   Input   Action
$i      +i$     Shift
$E      +i$     Reduce : E --> i
$E+     i$      Shift
$E+i    $       Shift
$E+E    $       Reduce : E --> i
$E      $       Reduce : E --> E + E
$       $
Finished parsing
```

8.6 Result

Implemented the program to do precedence parsing. It was compiled using g++ version 9.3.0, and executed in Ubuntu 20.04 and the above output was obtained.

Exp 9

9 First and Follow

9.1 Aim

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

9.2 Theory

First

FIRST is applied to the r.h.s. of a production rule, and tells us all the terminal symbols that can start sentences derived from that r.h.s.

Follow

FOLLOW is used only if the current non-terminal can derive ε; then we're interested in what could have followed it in a sentential form. (NB: A string can derive ε if and only if ε is in its FIRST set.

9.3 Algorithm

Algorithm 13: Algorithm for First and Follow

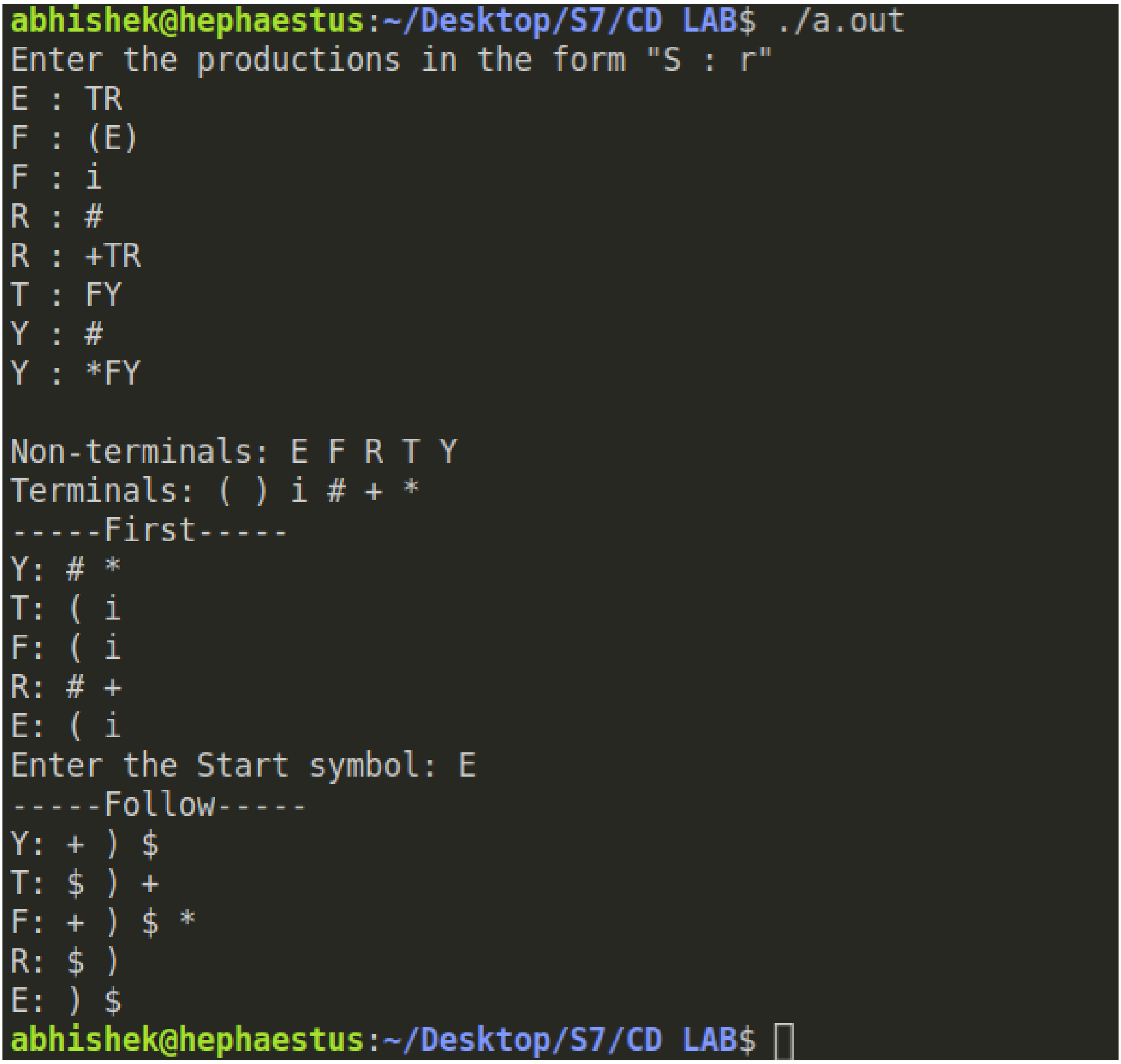
```
FIRST ( X ) for all grammar symbols X
    If X is terminal , FIRST ( X ) = { X }.
    If X -> e is a production , then add e to FIRST ( X ) .
    If X is a non - terminal , and X -> Y1 Y2 ... Yk is a production , and e is in all of FIRST ( Y1 ) , ... , FIRST ( Yk ) , then add e to FIRST ( X ) .
    If X is a non - terminal , and X -> Y1 Y2 ... Yk is a production , then add a to FIRST ( X ) if for some i , a is in FIRST ( Yi ) , and e is in all of FIRST ( Y1 ) ,... , FIRST ( Yi -1) .
FOLLOW ( A ) for all non - terminals A
    If $ is the input end - marker , and S is the start symbol , $ e FOLLOW ( S ) .
    If there is a production , A -> aBb , then ( FIRST ( b ) - e ) subset of FOLLOW ( B ) .
    If there is a production , A -> aB , or a production A -> aBb , where e belongs to FIRST ( b ) , then FOLLOW ( A ) subset of FOLLOW ( B ) .
```

9.4 Code

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<vector<string>> get_production(unordered_map<char, int> &non_term, int *num)
{
    int non = -1;
    string s;
    vector<vector<string>> production(100);
    getline(cin, s);
    while (s != "")
    {
        int non_index;
        char left = s[0];
        if (non_term.find(s[0]) == non_term.end())
        {
            non_term[s[0]] = ++non;
            non_index = non;
        }
        else
        {
            non_index = non_term[s[0]];
        }
        string right = s.substr(4, s.size() - 4);
        production[non].push_back(right);
        //cout << "right side " << right << endl;
        getline(cin, s);
    }
    *num = non;
    return production;
}

unordered_set<char> split_string(string s)
{
    int n = s.size();
    unordered_set<char> result;
    for (int i = 0; i < n; ++i)
    {
        if (s[i] != ' ')
        {
            result.insert(s[i]);
        }
    }
    return result;
}

vector<char> find_first(char c, vector<vector<string>> production, vector<vector<char>> &First, unordered_map<char, int> umap)
{
    vector<char> res;
    if (umap.find(c) == umap.end())
    {
        res.push_back(c);
        return res;
    }
    int num = umap[c];
    if (First[num].size() != 0)
    {
        return First[num];
    }
    int n = production[num].size();
    for (int i = 0; i < n; ++i) // iterate through each production
    {
        string m = production[num][i];
        int right_size = m.size();
        for (int j = 0; j < right_size; ++j) // iterate through each charecter in production
        {
            if (umap.find(m[j]) == umap.end()) //if right side of production  is a terminal
            {
                if (find(res.begin(), res.end(), m[j]) == res.end())
                {
                    res.push_back(m[j]);
                }

                break;
            }
            else // Non terminal
            {
                vector<char> temp = find_first(m[j], production, First, umap); // finding first of  j th non terminal
                //cout << "called for first of " << m[j] << endl;
                int first_char = temp.size();
                int flag = 1;
                for (int k = 0; k < first_char; ++k)
                {
                    if (temp[k] == '#')
                    {
                        // cout << "Epsilon found in first of " << m[j] << endl;
                        flag = 0;
                    }
                    if (find(res.begin(), res.end(), temp[k]) == res.end())
                    {
                        if (temp[k] != '#')
                        {
                            res.push_back(temp[k]);
                        }
                        else
                        {
                            if (j == right_size - 1)
                            {
                                res.push_back(temp[k]);
                            }
                        }
                    }
                }
                if (flag == 1)
                {
                    break;
                }
            }
        }
    }
    First[num] = res;
    return res;
}
unordered_set<char> find_follow(char c, vector<vector<string>> production, vector<unordered_set<char>> &follow, unordered_map<char, int> umap, vector<vector<char>> first)
{
    if (!follow[umap[c]].empty())
    {
        return follow[umap[c]];
    }
    //cout << "called follow of " << c << endl;
    unordered_set<char> res;
    if (umap[c] == 0)
    {
        //cout << "added $ in follow of " << c << endl;
        res.insert('$');
    }
    int n = production.size();
    for (int i = 0; i < n; ++i)
    {
        for (auto x : production[i])
        {                     // considering each production
            int m = x.size(); //read rhs charecter by charecter
            for (int j = 0; j < m; ++j)
            {
                if (x[j] == c) // if we find charecter in right side of production
                {
                    //cout << c << " found in production " << x << endl;
                    if (j == m - 1)
                    { //last element
                        //  cout << c << " is the edning charecter" << endl;
                        char check;
                        for (auto y : umap)
                        {
                            if (y.second == i)
                            {
                                check = y.first;
                            }
                        }
                        if (check != c)
                        {
                            unordered_set<char> sample = find_follow(check, production, follow, umap, first);
                            for (auto y : sample)
                            {
                                //cout << y << " inserted in follow of " << c << endl;

                                res.insert(y);
                            }
                            //cout << endl;
                        }
                    }
                    else
                    {
                        for (int k = j + 1; k < m; ++k)
                        {
                            int flag = 0;
                            if (umap.find(x[k]) == umap.end())
                            { // checking whether char is termi if so add and stop
                                // cout << "since found non terminal " << x[k] << "stop here added it "
                                //<< "in follow of " << c << endl;
                                res.insert(x[k]);
                                flag = 1;
                            }
                            else
                            { // if it is a non terminal then add its first
                                int first_b = first[umap[x[k]]].size();
                                for (int l = 0; l < first_b; ++l)
                                {
                                    if (first[umap[x[k]]][l] != '#')
                                    {
                                        res.insert(first[umap[x[k]]][l]);
                                        //cout << first[umap[x[k]]][l] << " Added to follow of " << c << endl;
                                        if (l == first_b - 1) // first[b] has #
                                        {
                                            char check;
                                            for (auto y : umap)
                                            {
                                                if (y.second == i)
                                                {
                                                    check = y.first;
                                                }
                                            }
                                            if (check != c)
                                            {
                                                unordered_set<char> sample = find_follow(check, production, follow, umap, first);
                                                for (auto y : sample)
                                                {
                                                    //cout << y << "added to follow of " << c << endl;
                                                    res.insert(y);
                                                }
                                            }
                                        }
                                    }
                                    else
                                    {
                                        flag = 1;
                                    }
                                }
                            }
                            if (flag == 1)
                            {
                                break;
                            }
                        }
                    }
                }
            }
        }
    }
    return res;
}
int main()
{
    int non = -1;
    string s;
    vector<vector<string>> production(100);
    cout << "Enter the productions in the form \"S : r\" " << endl;
    unordered_map<char, int> non_term;
    production = get_production(non_term, &non);
    unordered_set<char> terminals;
    unordered_set<char> non_terminals;
    cout << "Non-terminals: ";
    getline(cin, s);
    non_terminals = split_string(s);
    // for (auto x : non_terminals)
    // {
    //     cout << x << " ";
    // }
    // cout << endl;
    cout << "Terminals: ";
    getline(cin, s);
    vector<vector<char>> First(non + 1);
    terminals = split_string(s);
    // for (auto x : terminals)
    // {
```

```cpp
    //       cout << x << " ";
    // }
    // cout << endl;
    // cout << "number of non term is : " << non << endl;
    for (auto x : non_terminals)
    {
        First[non_term[x]] = find_first(x, production, First, non_term);
    }
    cout << "-----First----- " << endl;
    for (auto x : non_terminals)
    {
        cout << x << ": ";
        for (auto y : First[non_term[x]])
        {
            cout << y << " ";
        }
        cout << endl;
    }
    vector<unordered_set<char>> follow(non + 1);
    cout << "Enter the Start symbol: ";
    char c;
    cin >> c;
    cout << "-----Follow-----" << endl;
    for (auto x : non_terminals)
    {
        if (follow[non_term[x]].empty())
            follow[non_term[x]] = find_follow(x, production, follow, non_term, First);
    }
    for (auto x : non_terminals)
    {
        cout << x << ": ";
        for (auto y : follow[non_term[x]])
        {
            cout << y << " ";
        }
        cout << endl;
    }

    return 0;
}
```

9.5 Output



```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the productions in the form "S : r"
E : TR
F : (E)
F : i
R : #
R : +TR
T : FY
Y : #
Y : *FY

Non-terminals: E F R T Y
Terminals: ( ) i # + *
-----First-----
Y: # *
T: ( i
F: ( i
R: # +
E: ( i
Enter the Start symbol: E
-----Follow-----
Y: + ) $
T: $ ) +
F: + ) $ *
R: $ )
E: ) $
abhishek@hephaestus:~/Desktop/S7/CD LAB$
```

```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the productions in the form "S : r"
E : TR
F : (E)
F : i
R : #
R : +TR
T : FY
Y : #
Y : *FY

Non-terminals: E F R T Y
Terminals: ( ) i # + *
-----First-----
Y: # *
T: ( i
F: ( i
R: # +
E: ( i
Enter the Start symbol: E
-----Follow-----
Y: + ) $
T: $ ) +
F: + ) $ *
R: $ )
E: ) $
```

9.6 Result

Implemented the program to find FIRST and FOLLOW. It was compiled using g++ version 9.3.0, and executed in Ubuntu 20.04 and the above output was obtained.

Exp 10

10 Recursive descent parser

10.1 Aim

Construct a recursive descent parser for an expression.

10.2 Theory

Recursive Descent Parser

It is a kind of Top-Down Parser. A top-down parser builds the parse tree from the top to down, starting with the start non-terminal. A Predictive Parser is a special case of Recursive Descent Parser, where no Back Tracking is required. By carefully writing a grammar means eliminating left recursion and left factoring from it, the resulting grammar will be a grammar that can be parsed by a recursive descent parser.

10.3 Algorithm

Algorithm 14: Algorithm for Recursive descent parser

```
One parse method per non - terminal symbol
A non - terminal symbol on the right - hand side of a rewrite rule leads to a call to the
    parse method for that non - terminal
A terminal symbol on the right - hand side of a rewrite rule leads to " consuming " that token
     from the input token string
| in the CFG leads to " if - else " in the parser
```

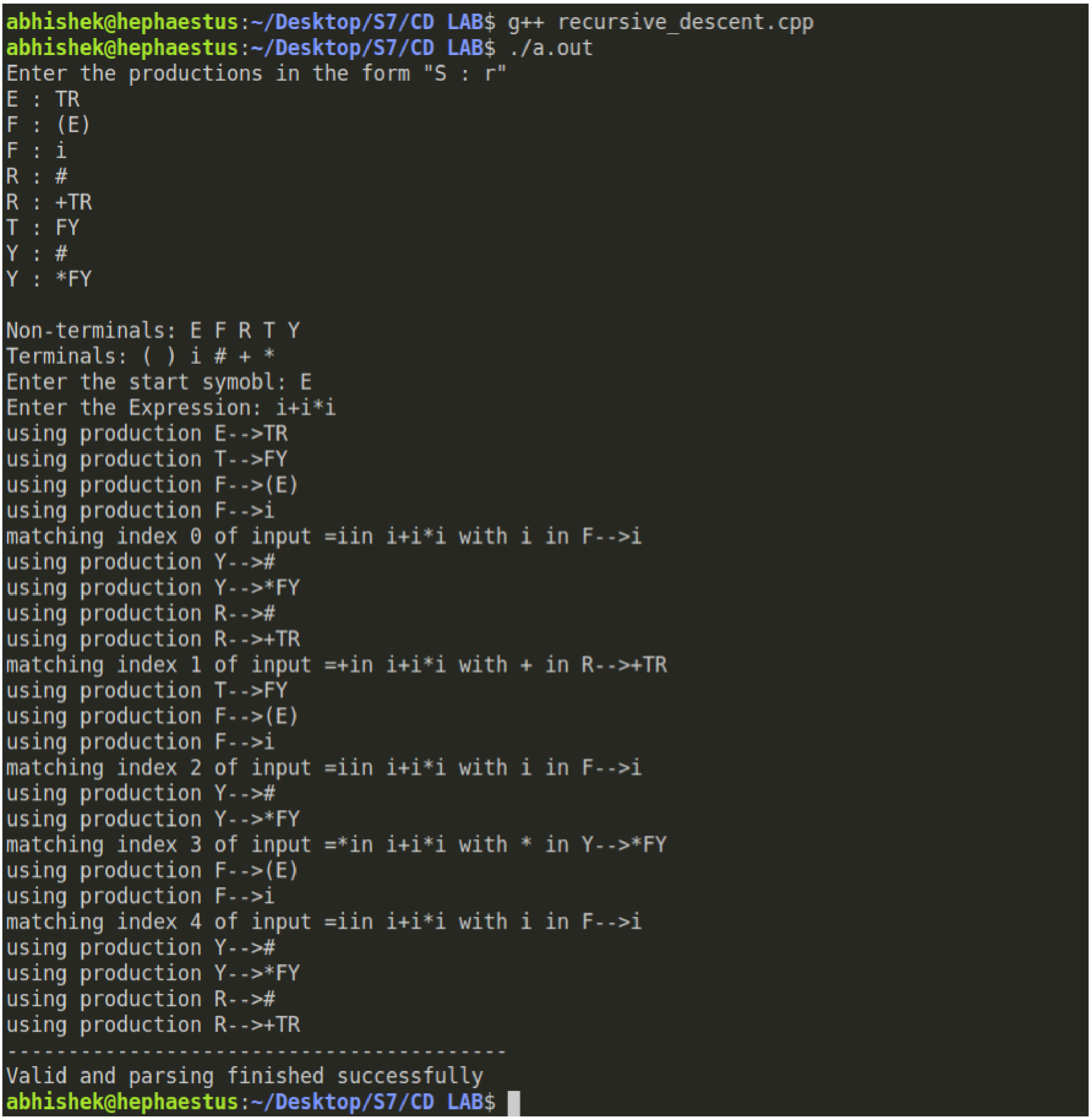
10.4 Code

```
#include <bits/stdc++.h>
using namespace std;

vector<vector<string>> get_production(unordered_map<char, int> &non_term, int *num)
{
    int non = -1;
    string s;
    vector<vector<string>> production(100);
    getline(cin, s);
    while (s != "")
    {
        int non_index;
        char left = s[0];
        if (non_term.find(s[0]) == non_term.end())
        {
            non_term[s[0]] = ++non;
            non_index = non;
        }
        else
        {
            non_index = non_term[s[0]];
        }
        string right = s.substr(4, s.size() - 4);
        production[non].push_back(right);
        //cout << "right side " << right << endl;
        getline(cin, s);
    }
    *num = non;
    return production;
}

unordered_set<char> split_string(string s)
```

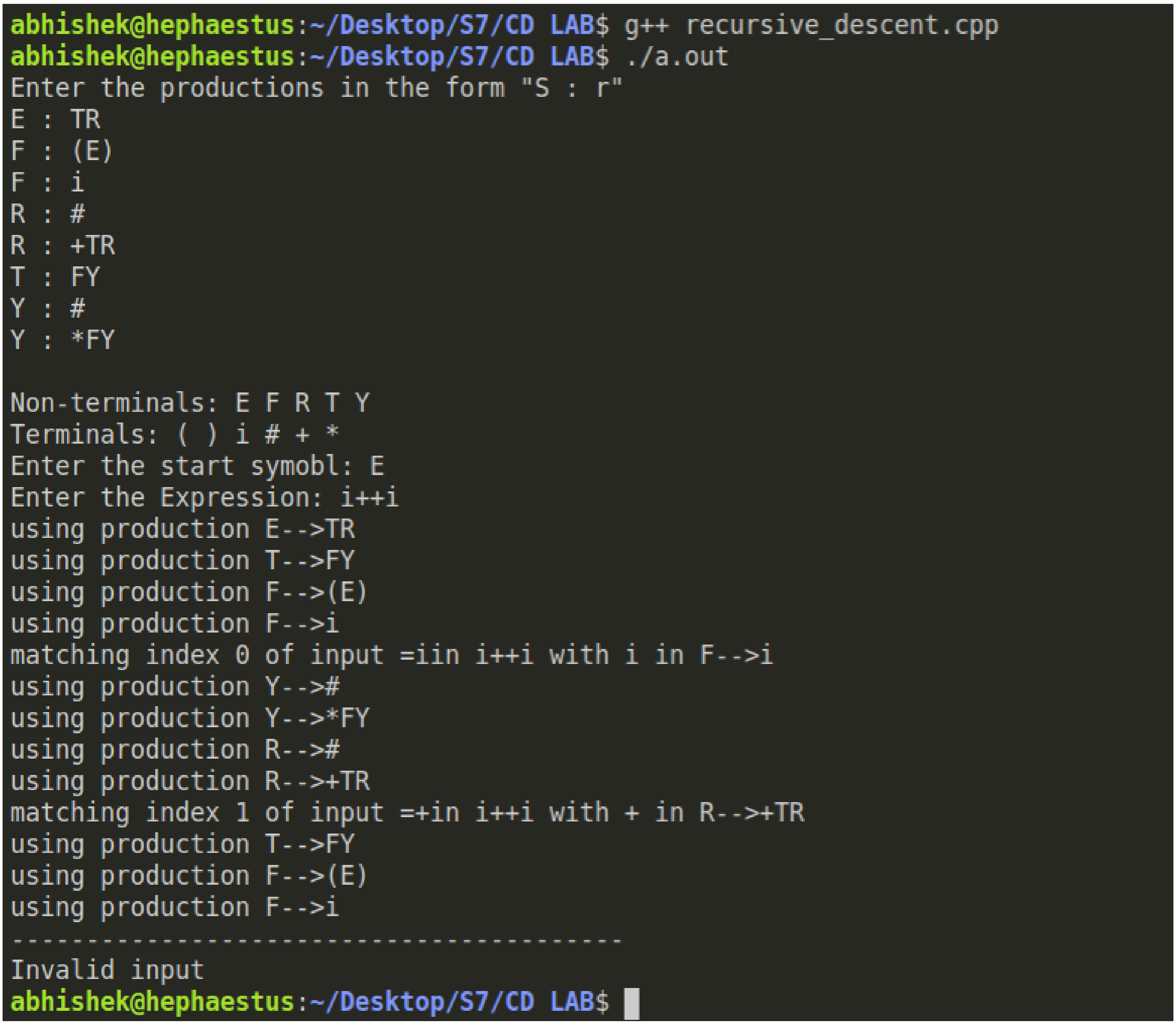
```cpp
{
    int n = s.size();
    unordered_set<char> result;
    for (int i = 0; i < n; ++i)
    {
        if (s[i] != ' ')
        {
            result.insert(s[i]);
        }
    }
    return result;
}

int method(vector<vector<string>> production, unordered_map<char, int> non_term, string input, int count, string crnt_prod, char E)
{
    int success = 0;
    cout << "using production " << E << "-->" << crnt_prod << endl;
    cout << "inspecting " << count << "th char in input" << endl;
    int size = crnt_prod.size();
    for (int i = 0; i < size; ++i)
    {
        if (non_term.find(crnt_prod[i]) == non_term.end())
        {
            if (crnt_prod[i] != input[count])
            {
                if (crnt_prod[i] == '#')
                {
                    cout << "epsilon found" << endl;
                    continue;
                }

                return -1;
            }
            else
            {
                cout << "matching index " << count << " of input =" << input[count] << " with " << crnt_prod[i] << " in " << crnt_prod << endl;
                success++;
            }
        }
        else
        {
            int fount = 0;
            char temp = crnt_prod[i];
            int non_term_num = non_term[crnt_prod[i]];
            for (int j = 0; j < production[non_term_num].size(); ++j)
            {
                int res = method(production, non_term, input, count + success, production[non_term_num][j], temp);
                if (res == 0)
                {
                    fount = 1;
                    continue;
                }
                if (res != -1)
                {
                    fount = 1;
                    success += res;
                    break;
                }
                else
                {
                    //contines loop
                }
            }
            if (fount == 0)
            {
                return -1;
            }
        }
    }

    return success;
}

```

```cpp
bool recursive_descent(vector<vector<string>> production, unordered_map<char, int> non_term,
    char E, string input, int count)
{
    int size = input.size();
    bool res = false;
    int non_term_num = non_term[E];
    for (int i = 0; i < production[non_term_num].size(); ++i)
    {
        int ans = method(production, non_term, input, count, production[non_term_num][i], E);
        if (ans != -1)
        {
            if (ans >= size)
            {
                cout << "-----------------------------------------" << endl;
                cout << "Valid and parsing finished successfully" << endl;
                return true;
            }
        }
    }
    cout << "-----------------------------------------" << endl;
    cout << "Invalid input" << endl;
    return false;
}

int main()
{
    int non = -1;
    string s;
    vector<vector<string>> production(100);
    cout << "Enter the productions in the form \"S : r\" " << endl;
    unordered_map<char, int> non_term;
    production = get_production(non_term, &non);
    unordered_set<char> terminals;
    unordered_set<char> non_terminals;
    cout << "Non-terminals: ";
    getline(cin, s);
    non_terminals = split_string(s);

    cout << "Terminals: ";
    getline(cin, s);
    terminals = split_string(s);

    char start;
    cout << "Enter the start symobl: ";
    cin >> start;
    cout << "Enter the Expression: ";
    cin >> s;
    bool val = recursive_descent(production, non_term, start, s, 0);

    return 0;
}
```

10.5 Output



```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ g++ recursive_descent.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the productions in the form "S : r"
E : TR
F : (E)
F : i
R : #
R : +TR
T : FY
Y : #
Y : *FY

Non-terminals: E F R T Y
Terminals: ( ) i # + *
Enter the start symobl: E
Enter the Expression: i+i*i
using production E-->TR
using production T-->FY
using production F-->(E)
using production F-->i
matching index 0 of input =iin i+i*i with i in F-->i
using production Y-->#
using production Y-->*FY
using production R-->#
using production R-->+TR
matching index 1 of input =+in i+i*i with + in R-->+TR
using production T-->FY
using production F-->(E)
using production F-->i
matching index 2 of input =iin i+i*i with i in F-->i
using production Y-->#
using production Y-->*FY
matching index 3 of input =*in i+i*i with * in Y-->*FY
using production F-->(E)
using production F-->i
matching index 4 of input =iin i+i*i with i in F-->i
using production Y-->#
using production Y-->*FY
using production R-->#
using production R-->+TR
-----------------------------------------
Valid and parsing finished successfully
abhishek@hephaestus:~/Desktop/S7/CD LAB$
```



```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ g++ recursive_descent.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the productions in the form "S : r"
E : TR
F : (E)
F : i
R : #
R : +TR
T : FY
Y : #
Y : *FY

Non-terminals: E F R T Y
Terminals: ( ) i # + *
Enter the start symobl: E
Enter the Expression: i++i
using production E-->TR
using production T-->FY
using production F-->(E)
using production F-->i
matching index 0 of input =iin i++i with i in F-->i
using production Y-->#
using production Y-->*FY
using production R-->#
using production R-->+TR
matching index 1 of input =+in i++i with + in R-->+TR
using production T-->FY
using production F-->(E)
using production F-->i
----------------------------------------
Invalid input
abhishek@hephaestus:~/Desktop/S7/CD LAB$ 
```

```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ g++ recursive_descent.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the productions in the form "S : r"
E : TR
F : (E)
F : i
R : #
R : +TR
T : FY
Y : #
Y : *FY

Non-terminals: E F R T Y
Terminals: ( ) i # + *
Enter the start symobl: E
Enter the Expression: i+i*i
using production E-->TR
using production T-->FY
using production F-->(E)
using production F-->i
matching index 0 of input =iin i+i*i with i in F-->i
using production Y-->#
using production Y-->*FY
using production R-->#
using production R-->+TR
matching index 1 of input =+in i+i*i with + in R-->+TR
using production T-->FY
using production F-->(E)
using production F-->i
matching index 2 of input =iin i+i*i with i in F-->i
using production Y-->#
using production Y-->*FY
matching index 3 of input =*in i+i*i with * in Y-->*FY
using production F-->(E)
using production F-->i
matching index 4 of input =iin i+i*i with i in F-->i
using production Y-->#
using production Y-->*FY
using production R-->#
using production R-->+TR
----------------------------------------
Valid and parsing finished successfully
abhishek@hephaestus:~/Desktop/S7/CD LAB$ g++ recursive_descent.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the productions in the form "S : r"
E : TR
F : (E)
F : i
R : #
R : +TR
T : FY
Y : #
Y : *FY

Non-terminals: E F R T Y
Terminals: ( ) i # + *
Enter the start symobl: E
Enter the Expression: i++i
using production E-->TR
using production T-->FY
using production F-->(E)
using production F-->i
matching index 0 of input =iin i++i with i in F-->i
using production Y-->#
using production Y-->*FY
using production R-->#
using production R-->+TR
matching index 1 of input =+in i++i with + in R-->+TR
using production T-->FY
using production F-->(E)
using production F-->i
----------------------------------------
Invalid input
```

10.6 Result

Implemented the program to construct a recursive descent parser. It was compiled using g++ version 9.3.0, and executed in Ubuntu 20.04 and the above output was obtained.

Exp 11

11 Shift Reduce Parser

11.1 Aim

Construct a Shift Reduce Parser for a given language.

11.2 Theory

Shift Reduce parser

Shift Reduce parser attempts for the construction of parse in a similar manner as done in bottom up parsing i.e. the parse tree is constructed from leaves(bottom) to the root(up). A more general form of shift reduce parser is LR parser.
This parser requires some data structures i.e.

- A input buffer for storing the input string.
- A stack for storing and accessing the production rules.

Basic Operations –

1. **Shift**: This involves moving of symbols from input buffer onto the stack.
2. **Reduce:** If the handle appears on top of the stack then, its reduction by using appropriate production rule is done i.e. RHS of production rule is popped out of stack and LHS of production rule is pushed onto the stack.
3. **Accept:** If only start symbol is present in the stack and the input buffer is empty then, the parsing action is called accept. When accept action is obtained, it is means successful parsing is done.
4. **Error:** This is the situation in which the parser can neither perform shift action nor reduce action and not even accept action.

11.3 Algorithm

Algorithm 15: Algorithm for Shift Reduce parser

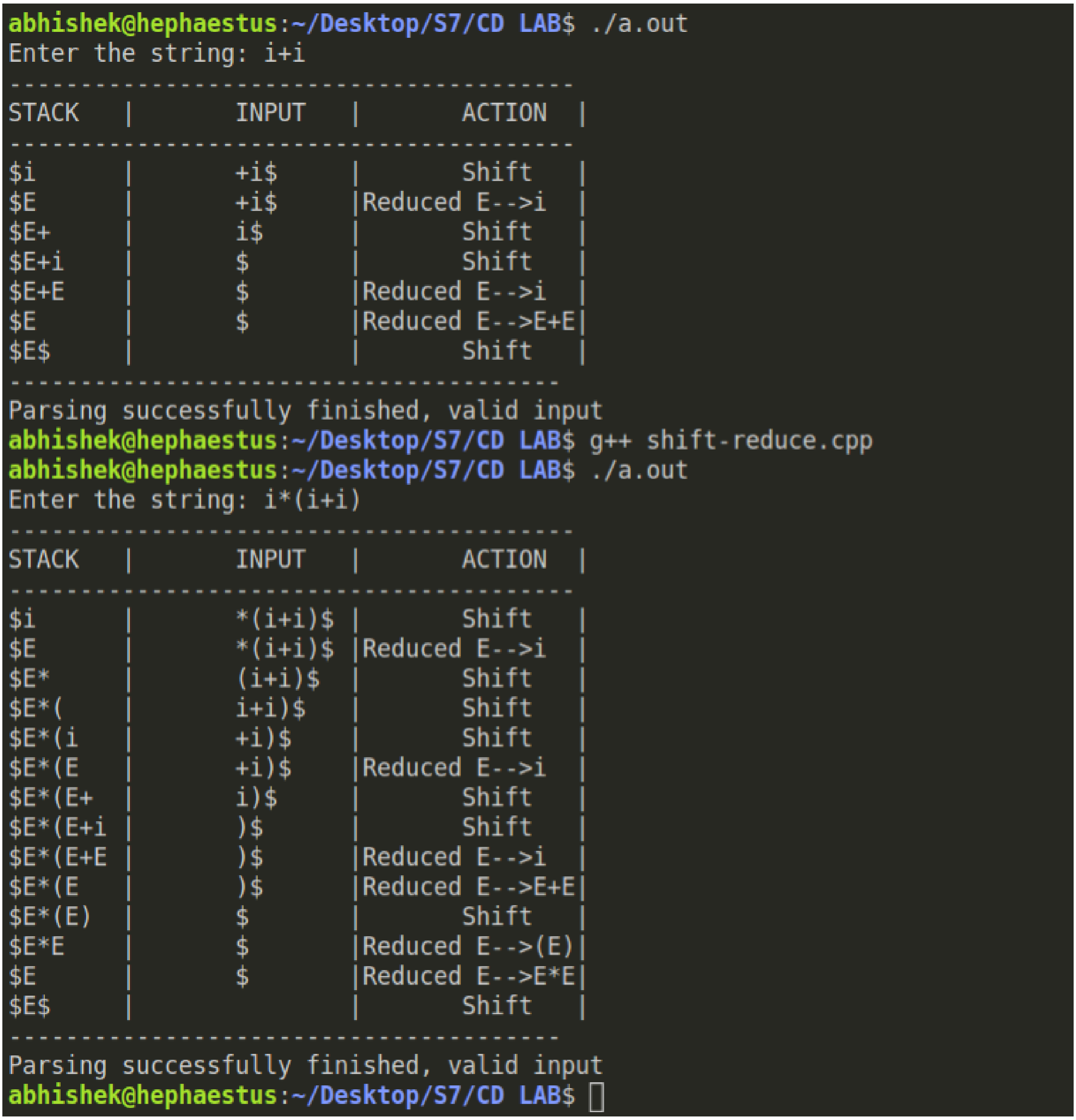
```
loop forever :
    for top - of - stack symbol , s , and next input symbol , a case
    action of T [s , a ]
        shift x : ( x is a STATE number )
            push a , then x on the top of the stack and
            advance ip to point to the next input symbol .
        reduce y : ( y is a PRODUCTION number )
            Assume that the production is of the form
                A == > beta
            pop 2 * | beta | symbols of the stack . At this
            point the top of the stack should be a state number ,
            say s      , push A , then goto of T [ s      ,A ] ( a state number )
            on the top of the stack . Output the production
                A == > beta .
        accept :
            return --- a successful parse .
        default :
            error --- the input string is not in the language .
```

11.4 Code

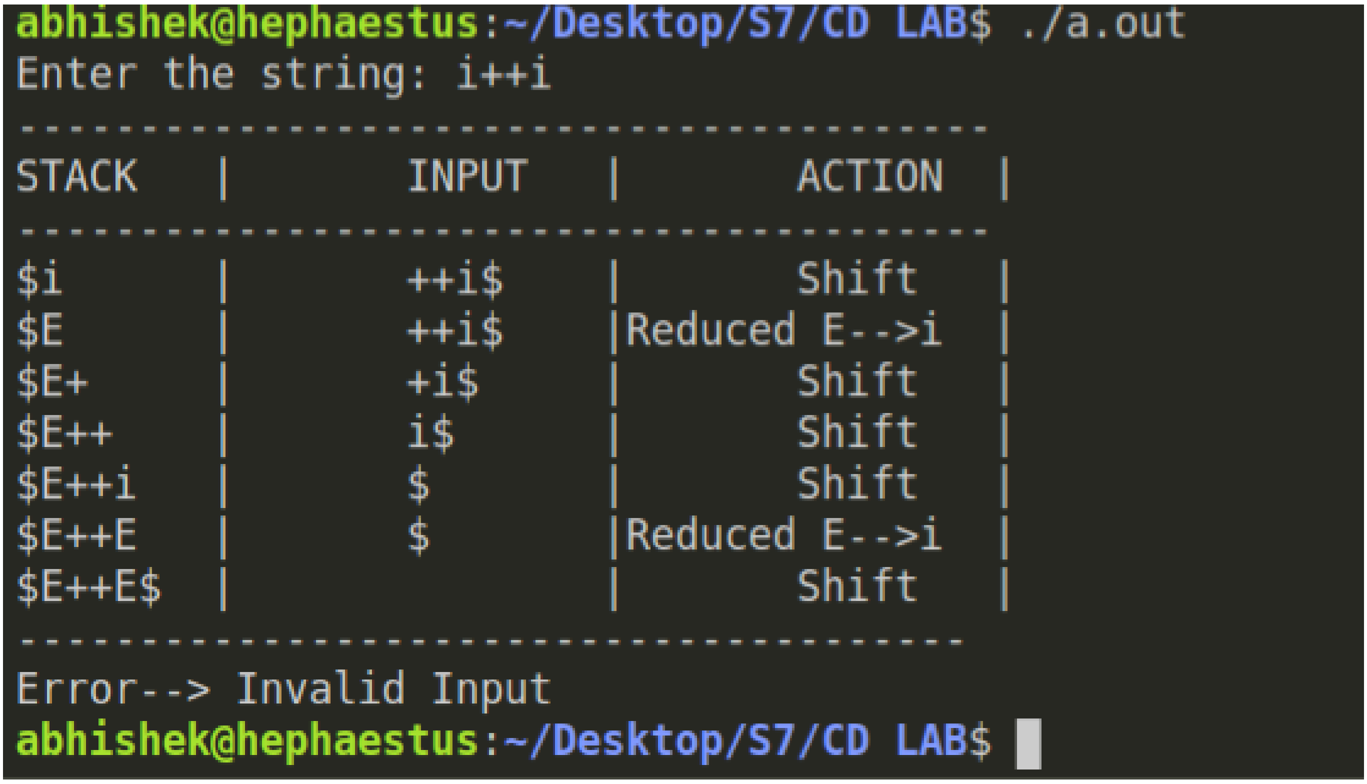
```cpp
#include <bits/stdc++.h>
using namespace std;
void print_stack(stack<char> check)
{
    string s = "";
    while (!check.empty())
    {
        s = check.top() + s;
        check.pop();
    }
    cout << s;
}
string last_3(stack<char> check)
{
    string res = "";
    for (int i = 0; i < 3; ++i)
    {
        res = check.top() + res;
        check.pop();
    }
    return res;
}

int main()
{
    vector<char> lhs = {'E'};
    unordered_set<string> rhs = {"E+E", "(E)", "i", "E*E"};
    cout << "Enter the string: ";
    string s;
    cin >> s;
    s += "$";
    int n = s.size(), count = 1, i = 0;
    stack<char> SR;
    char a, b, c;
    SR.push('$');
    cout << "----------------------------------------" << endl;
    cout << "STACK\t|\tINPUT\t|\tACTION\t|" << endl;
    cout << "----------------------------------------" << endl;
    while (true)
    {
        if (count >= 3)
        {
            string over = last_3(SR);
            //cout << "string found is " << over << endl;
            if (over == "$E$")
            {
                cout << "---------------------------------------" << endl;
                cout << "Parsing successfully finished, valid input" << endl;
                break;
            }
            if (rhs.find(over) != rhs.end())
            {
                SR.pop();
                SR.pop();
                SR.pop();
                SR.push('E');
                print_stack(SR);
                cout << "\t|\t";
                cout << s.substr(i, n - i) << "\t|Reduced E-->" << over << "|" << endl;
                // cout << "---------------------------------------" << endl;
                count -= 2;
                continue;
            }
        }
        if (SR.top() == 'i')
        {
            SR.pop();
            SR.push('E');
            print_stack(SR);
            cout << "\t|\t";
            cout << s.substr(i, n - i) << "\t|Reduced E-->i\t|" << endl;
            //cout << "---------------------------------------" << endl;
            continue;
        }
        if (i >= n)
        {
            cout << "----------------------------------------" << endl;
            cout << "Error--> Invalid Input" << endl;
            break;
        }
        SR.push(s[i]);
        print_stack(SR);
        cout << "\t|\t";
        cout << s.substr(i + 1, n - i) << "\t|\tShift\t|" << endl;
        //cout << "----------------------------------------" << endl;
        count++;
        i++;
    }
    return 0;
}
```

Code for SR Parser

11.5 Output



```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the string: i+i
----------------------------------------
STACK   |       INPUT   |       ACTION  |
----------------------------------------
$i      |       +i$     |       Shift   |
$E      |       +i$     |Reduced E-->i  |
$E+     |       i$      |       Shift   |
$E+i    |       $       |       Shift   |
$E+E    |       $       |Reduced E-->i  |
$E      |       $       |Reduced E-->E+E|
$E$     |               |       Shift   |
---------------------------------------
Parsing successfully finished, valid input
abhishek@hephaestus:~/Desktop/S7/CD LAB$ g++ shift-reduce.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the string: i*(i+i)
----------------------------------------
STACK   |       INPUT   |       ACTION  |
----------------------------------------
$i      |       *(i+i)$ |       Shift   |
$E      |       *(i+i)$ |Reduced E-->i  |
$E*     |       (i+i)$  |       Shift   |
$E*(    |       i+i)$   |       Shift   |
$E*(i   |       +i)$    |       Shift   |
$E*(E   |       +i)$    |Reduced E-->i  |
$E*(E+  |       i)$     |       Shift   |
$E*(E+i |       )$      |       Shift   |
$E*(E+E |       )$      |Reduced E-->i  |
$E*(E   |       )$      |Reduced E-->E+E|
$E*(E)  |       $       |       Shift   |
$E*E    |       $       |Reduced E-->(E)|
$E      |       $       |Reduced E-->E*E|
$E$     |               |       Shift   |
---------------------------------------
Parsing successfully finished, valid input
abhishek@hephaestus:~/Desktop/S7/CD LAB$ 
```



```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the string: i++i
---------------------------------------
STACK    |       INPUT    |       ACTION   |
---------------------------------------
$i       |       ++i$     |       Shift    |
$E       |       ++i$     |Reduced E-->i   |
$E+      |       +i$      |       Shift    |
$E++     |       i$       |       Shift    |
$E++i    |       $        |       Shift    |
$E++E    |       $        |Reduced E-->i   |
$E++E$   |                |       Shift    |
--------------------------------------
Error--> Invalid Input
abhishek@hephaestus:~/Desktop/S7/CD LAB$
```

```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ g++ shift-reduce.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the string: i+i
---------------------------------------
STACK   |       INPUT   |       ACTION  |
---------------------------------------
$i      |       +i$     |       Shift   |
$E      |       +i$     |Reduced E-->i  |
$E+     |       i$      |       Shift   |
$E+i    |       $       |       Shift   |
$E+E    |       $       |Reduced E-->i  |
$E      |       $       |Reduced E-->E+E|
$E$     |               |       Shift   |
--------------------------------------
Parsing successfully finished, valid input
abhishek@hephaestus:~/Desktop/S7/CD LAB$ g++ shift-reduce.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the string: i*(i+i)
---------------------------------------
STACK   |       INPUT   |       ACTION  |
---------------------------------------
$i      |       *(i+i)$ |       Shift   |
$E      |       *(i+i)$ |Reduced E-->i  |
$E*     |       (i+i)$  |       Shift   |
$E*(    |       i+i)$   |       Shift   |
$E*(i   |       +i)$    |       Shift   |
$E*(E   |       +i)$    |Reduced E-->i  |
$E*(E+  |       i)$     |       Shift   |
$E*(E+i |       )$      |       Shift   |
$E*(E+E |       )$      |Reduced E-->i  |
$E*(E   |       )$      |Reduced E-->E+E|
$E*(E)  |       $       |       Shift   |
$E*E    |       $       |Reduced E-->(E)|
$E      |       $       |Reduced E-->E*E|
$E$     |               |       Shift   |
--------------------------------------
```

```
Parsing successfully finished, valid input
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the string: i++i
----------------------------------------
STACK   |       INPUT   |       ACTION  |
----------------------------------------
$i      |       ++i$    |       Shift   |
$E      |       ++i$    |Reduced E-->i  |
$E+     |       +i$     |       Shift   |
$E++    |       i$      |       Shift   |
$E++i   |       $       |       Shift   |
$E++E   |       $       |Reduced E-->i  |
$E++E$  |               |       Shift   |
---------------------------------------
Error--> Invalid Input
abhishek@hephaestus:~/Desktop/S7/CD LAB$
```

11.6 Result

Implemented the program to construct a Shift Reduce parser. It was compiled using g++ version 9.3.0, and executed in Ubuntu 20.04 and the above output was obtained.

Exp 12

12 Loop Unrolling

12.1 Aim

Write a program to perform loop unrolling

12.2 Theory

Loop Unrolling

Loop unrolling is a loop transformation technique that helps to optimize the execution time of a program. We basically remove or reduce iterations. Loop unrolling increases the program's speed by eliminating loop control instruction and loop test instructions.

12.2.1 Advantages

- Increases program efficiency.
- Reduces loop overhead.
- If statements in loop are not dependent on each other, they can be executed in parallel.

12.2.2 Disadvantages

- Increased program code size, which can be undesirable.
- Possible increased usage of register in a single iteration to store temporary variables which may reduce performance.
- Apart from very small and simple codes, unrolled loops that contain branches are even slower than recursions.

12.3 Algorithm

Algorithm 16: Algorithm for Loop Unrolling

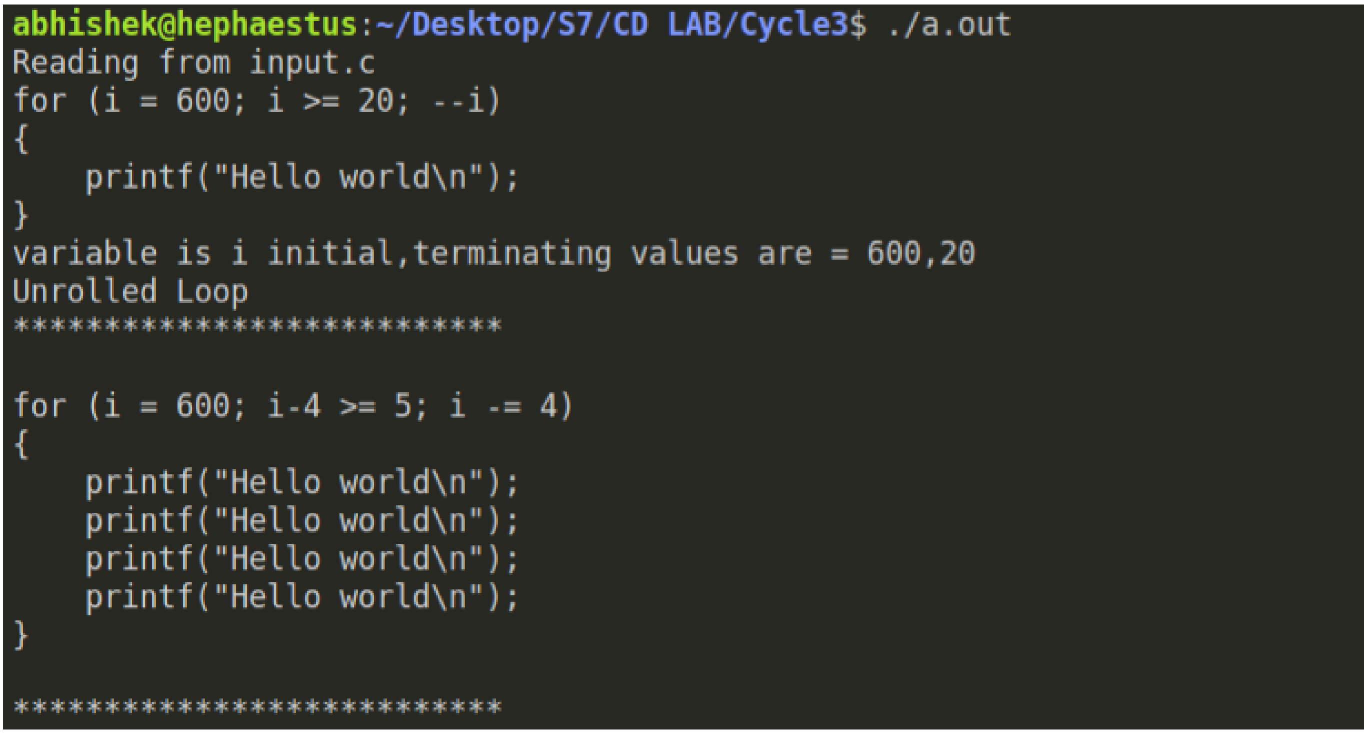
```
Read the loop
Store initial, terminal condition and variable
unroll the loop with modifying initial and terminal condition
change variable name accordingly
```

12.4 Code

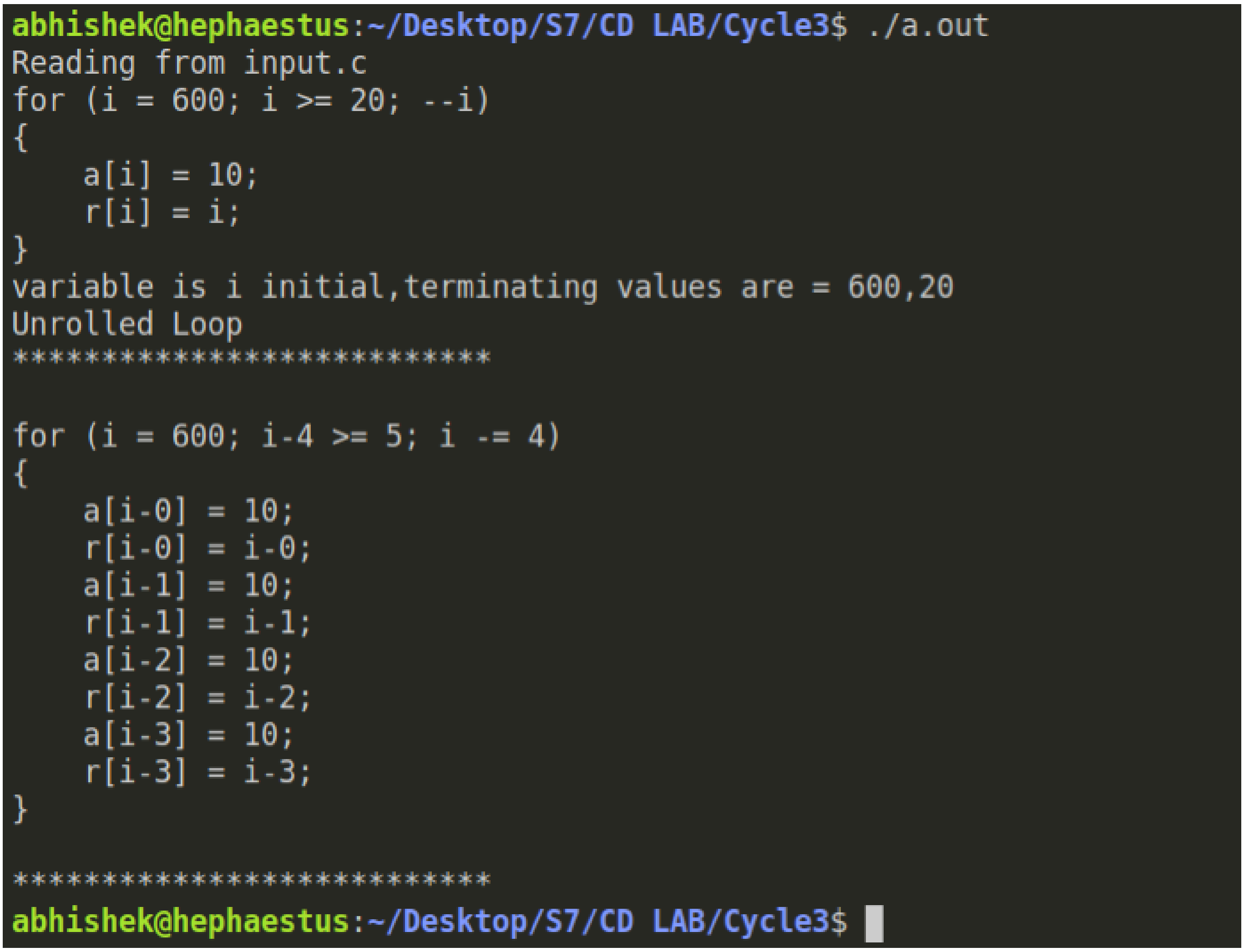
```cpp
#include <bits/stdc++.h>
using namespace std;

string beautify(string s) // to remove unneccessery space , () etc in the loop
{
    string new_s = "";
    int n = s.size();
    int flag = 0;
    for (int i = 1; i < n; ++i)
    {

        if (s[i - 1] == ’(’)
        {
            flag = 1;
        }
        else if (s[i] == ')')
        {
            break;
        }
        if (flag == 0)
        {
            continue;
        }
        if (s[i] != ' ')
        {
            new_s += s[i];
        }
    }
    return new_s;
}
void get_det(string s, int *start, int *end, int *cond, char *var, string *relop) // find the variable start and end condition etc
{
    s = beautify(s);
    //cout << s << " trimmed string " << endl;
    *var = s[0]; // variable returned
    int first = 0, second = 0, n = s.size();
    for (int i = 0; i < n; ++i)
    { // finding index of ;
        if (s[i] == ';')
        {
            first = second;
            second = i;
        }
    }
    string init = s.substr(2, first - 2);
    //cout << init << " initial value" << endl;
    *start = stoi(init);
    //cout << s[first + 2] << " " << s[first + 3] << endl;
    if (s.substr(first + 2, 2) == "<=")
    {
        *relop = "<=";
        init = s.substr(first + 4, second - first - 4);
        //cout << init << " terminal value" << endl;
    }
    else if (s[first + 2] == '<')
    {
        *relop = "<";
        init = s.substr(first + 3, second - first - 3);
        //cout << init << " terminal value" << endl;
    }
    else if (s.substr(first + 2, 2) == ">=")
    {
        *relop = ">=";
        init = s.substr(first + 4, second - first - 4);
        //cout << init << " terminal value" << endl;
    }
    else
    {
        *relop = ">";
        init = s.substr(first + 3, second - first - 3);
        //cout << init << " terminal value" << endl;
    }
    *end = stoi(init);
    if (s[second + 1] == '+')
    {
        *cond = 0;
    }
    else
    {
        *cond = 1;
    }
}
void print_with_newval(vector<string> lines, vector<pair<int, int>> variable, string replace)
{
    int rep_count = variable.size();
    int n = lines.size(), curr = 0, flag;
    if (curr == rep_count)
    {
        flag = 1;
    }
    else
    {
        flag = 0;
    }
    for (int i = 2; i < n - 1; ++i)
    {
        if (flag == 1 || variable[curr].first != i)
        { // print thr line if falg = 1 or the line is free of variable
            // cout << "no variable in line " << i << endl;
            cout << lines[i] << endl;
        }
        else
        {
            int pos = 0;
            while (variable[curr].first == i)
            { // repeat untill the line has the  loop variable
                //cout << "line found";
                cout << lines[i].substr(pos, variable[curr].second - pos);
                pos = variable[curr].second + 1;
                cout << replace;
                curr++;
                if (curr == rep_count)
                {
                    flag = 1;
                    break;
                }
            }
            if (pos < lines[i].size())
            { // print the rest of the line
                cout << lines[i].substr(pos, lines[i].size() - pos) << endl;
            }
        }
    }
}
int main()
{
    vector<string> lines;
    string s, relop;
    ifstream file("loop.c");
    cout << "Reading from input.c" << endl;
    while (getline(file, s))
    {
        cout << s << endl;
        lines.push_back(s);
    }
    int start, end, cond; //cond = 0 for < , 1 for <= , 2 for >,  3 for >=
    char var;
    get_det(lines[0], &start, &end, &cond, &var, &relop);
    cout << "variable is " << var << " initial,terminating values are = " << start << "," << end << endl;
    cout << "Unrolled Loop" << endl;
    cout << "***************************" << endl
         << endl;
    vector<pair<int, int>> variable;
    for (int i = 2; i < lines.size() - 1; ++i)
    {
        for (int j = 0; j < lines[i].size(); ++j)
        {
            if (lines[i][j] == var)
            {
                if (j == 0 && !isalnum(lines[i][j + 1]))
                {
                    variable.push_back({i, j});
                }
                else if (j == lines[i].size() - 1 && !isalnum(lines[i][j - 1]))
                {
                    variable.push_back({i, j});
                }
                else if (!isalnum(lines[i][j + 1]) && !isalnum(lines[i][j - 1]))
                {
                    variable.push_back({i, j});
                }
            }
        }
    }

    // for (auto x : variable)
    // {
    //     cout << x.first << " " << x.second << endl;
    // }
    string i_d;
    i_d = cond == 0 ? '+' : '-';
    cout << "for (" << var << " = " << start << "; " << var << i_d << "4"
         << " " << relop << " " << end / 4 << "; ";

    cout << var << " " << i_d << "= 4)" << endl;
    cout << "{" << endl;
    print_with_newval(lines, variable, var + i_d + '0');
    print_with_newval(lines, variable, var + i_d + '1');
    print_with_newval(lines, variable, var + i_d + '2');
    print_with_newval(lines, variable, var + i_d + '3');
    cout << "}" << endl
         << endl;
    cout << "***************************" << endl;
}
```

Code for SR Parser

12.5 Output



```
abhishek@hephaestus:~/Desktop/S7/CD LAB/Cycle3$ ./a.out
Reading from input.c
for (i = 600; i >= 20; --i)
{
    printf("Hello world\n");
}
variable is i initial,terminating values are = 600,20
Unrolled Loop
***************************

for (i = 600; i-4 >= 5; i -= 4)
{
    printf("Hello world\n");
    printf("Hello world\n");
    printf("Hello world\n");
    printf("Hello world\n");
}

***************************
```



```
abhishek@hephaestus:~/Desktop/S7/CD LAB/Cycle3$ ./a.out
Reading from input.c
for (i = 600; i >= 20; --i)
{
    a[i] = 10;
    r[i] = i;
}
variable is i initial,terminating values are = 600,20
Unrolled Loop
***************************

for (i = 600; i-4 >= 5; i -= 4)
{
    a[i-0] = 10;
    r[i-0] = i-0;
    a[i-1] = 10;
    r[i-1] = i-1;
    a[i-2] = 10;
    r[i-2] = i-2;
    a[i-3] = 10;
    r[i-3] = i-3;
}

***************************
abhishek@hephaestus:~/Desktop/S7/CD LAB/Cycle3$
```

```
abhishek@hephaestus:~/Desktop/S7/CD LAB/Cycle3$ ./a.out
Reading from input.c
for (i = 600; i >= 20; --i)
{
    printf("Hello world\n");
}
variable is i initial,terminating values are = 600,20
Unrolled Loop
***************************

for (i = 600; i-4 >= 5; i -= 4)
{
    printf("Hello world\n");
    printf("Hello world\n");
    printf("Hello world\n");
    printf("Hello world\n");
}

***************************
abhishek@hephaestus:~/Desktop/S7/CD LAB/Cycle3$ g++ loop_unroll.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB/Cycle3$ ./a.out
Reading from input.c
for (i = 600; i >= 20; --i)
{
    a[i] = 10;
    r[i] = i;
}
variable is i initial,terminating values are = 600,20
```

```
Unrolled Loop
***************************

for (i = 600; i-4 >= 5; i -= 4)
{
    a[i-0] = 10;
    r[i-0] = i-0;
    a[i-1] = 10;
    r[i-1] = i-1;
    a[i-2] = 10;
    r[i-2] = i-2;
    a[i-3] = 10;
    r[i-3] = i-3;
}

***************************
abhishek@hephaestus:~/Desktop/S7/CD LAB/Cycle3$
```

12.6 Result

Implemented the program for loop unrolling. It was compiled using g++ version 9.3.0, and executed in Ubuntu 20.04 and the above output was obtained.

Exp 13

13 Constant Propagation

13.1 Aim

Write a program to perform constant propagation.

13.2 Theory

Constant Propagation.

Expressions with constant operands can be evaluated at compile time, thus improving run-time performance and reducing code size by avoiding evaluation at compile-time. Constant propagation is the process of substituting the values of known constants in expressions at compile time. Such constants include those defined above, as well as intrinsic functions applied to constant values.

13.3 Algorithm

Algorithm 17: Algorithm for Constant propagation

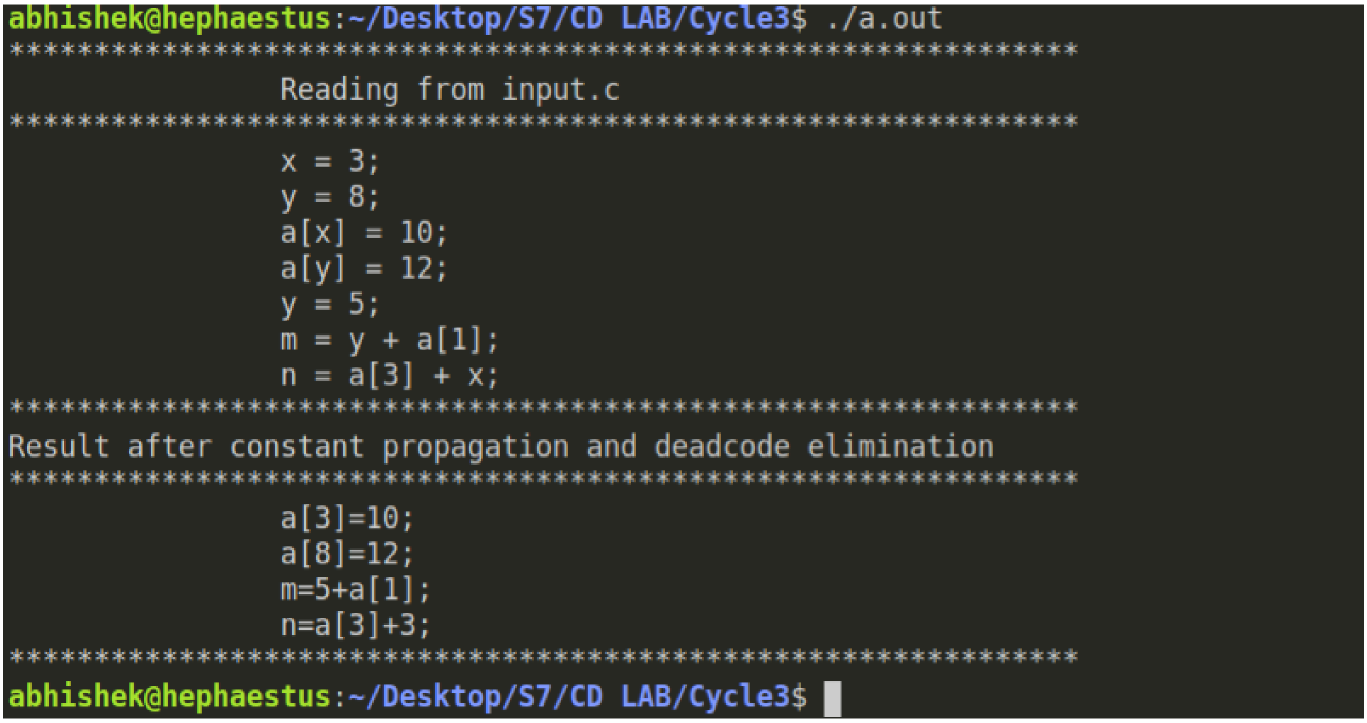
```
Start
For all statement in the program do begin
for each output v of s do valout ( v , s )=unknown
for each input w of s do
if w is a variable then valin(w,s)=unknown
else valin(w, s )= constant value of w
end
```

13.4 Code

```
#include <bits/stdc++.h>
using namespace std;
string beautify(string s) // to remove unneccessery space , () etc in the loop
{
    string new_s = "";
    int n = s.size();
    int flag = 0;
    for (int i = 0; i < n; ++i)
    {
        if (s[i] != ' ')
        {
            new_s += s[i];
        }
    }
    return new_s;
}
void print_star()
{
    cout << "****************************************************************" << endl;
}
bool is_id(string s, int i)
{
    if (!isalpha(s[i]))
    {
        return false;
    }
    if (i == 0)
    {
        if (!isalnum(s[i + 1]))
        {
            return true;
        }
    }
    else if (i == s.size() - 1)
    {
        if (!isalnum(s[i - 1]))
        {
            return true;
        }
    }
    else
    {
        if (!isalnum(s[i - 1]) && !isalnum(s[i + 1]))
        {
            return true;
        }
    }
    return false;
}
vector<string> constant(vector<string> lines, unordered_map<char, int> values)
{
    vector<string> result;
    int n = lines.size();
    for (int i = 0; i < n; ++i)
    {
        int len = lines[i].size();
        if (regex_match(lines[i], regex("[a-zA-z]=[0-9]*;")))
        {
            //cout << "true" << endl;
            char variable = lines[i][0];
            string data = lines[i].substr(2, n - 1);
            int cons = stoi(data);
            //cout << "variale is: " << variable << " value: " << cons << endl;
            values[variable] = cons;
        }
        else
        {
            string append = "";
            for (int j = 0; j < len; ++j)
            {
                if (is_id(lines[i], j))
                {
                    if (values.find(lines[i][j]) != values.end())
                    {
                        int cons = values[lines[i][j]];
                        string s = to_string(cons);
                        append += s;
                        //cout << "variable found and appending " << s << endl;
                    }
                    else
                    {
                        append += lines[i][j];
                        // cout << "variable found but not value and appending " << lines[i][j] << endl;
                    }
                }
                else
                {
                    append += lines[i][j];
                    //cout << "variable not found and appending " << lines[i][j] << endl;
                }
            }
            result.push_back(append);
            //cout << append << endl;
        }
    }
    return result;
}
int main()
{
    vector<string> lines;
    string s, temp;
    ifstream file("constant.c");
    print_star();
    cout << "\t\t"
         << "Reading from input.c" << endl;
    print_star();
    while (getline(file, s))
    {
        cout << "\t\t" << s << endl;
        s = beautify(s);
        lines.push_back(s);
    }
    unordered_map<char, int> values;
    vector<string> result = constant(lines, values);
    print_star();
    cout << "Result after constant propagation and deadcode elimination" << endl;
    print_star();
    for (auto x : result)
    {
        cout << "\t\t" << x << endl;
    }
    print_star();

    return 0;
}
```

Code for Constant Propagation

13.5 Output



```
abhishek@hephaestus:~/Desktop/S7/CD LAB/Cycle3$ ./a.out
***************************************************************
                Reading from input.c
***************************************************************
                x = 3;
                y = 8;
                a[x] = 10;
                a[y] = 12;
                y = 5;
                m = y + a[1];
                n = a[3] + x;
***************************************************************
Result after constant propagation and deadcode elimination
***************************************************************
                a[3]=10;
                a[8]=12;
                m=5+a[1];
                n=a[3]+3;
***************************************************************
abhishek@hephaestus:~/Desktop/S7/CD LAB/Cycle3$
```

```
abhishek@hephaestus:~/Desktop/S7/CD LAB/Cycle3$ ./a.out
***************************************************************
                Reading from input.c
***************************************************************
                x = 3;
                y = 8;
                a[x] = 10;
                a[y] = 12;
                y = 5;
                m = y + a[1];
                n = a[3] + x;
***************************************************************
Result after constant propagation and deadcode elimination
***************************************************************
                a[3]=10;
                a[8]=12;
                m=5+a[1];
                n=a[3]+3;
***************************************************************
```

13.6 Result

Implemented the program for constant propagation. It was compiled using g++ version 9.3.0, and executed in Ubuntu 20.04 and the above output was obtained.

Exp 14

14 Intermediate Code Generation

14.1 Aim

Implement Intermediate code generation for simple expressions

14.2 Theory

Intermediate Code Generation.

In the analysis-synthesis model of a compiler, the front end of a compiler translates a source program into an independent intermediate code, then the back end of the compiler uses this intermediate code to generate the target code (which can be understood by the machine).

Intermediate code can be either language specific (e.g., Bytecode for Java) or language. independent (three-address code).

The following are commonly used intermediate code representation:

1. Postfix Notation.
2. Three-Address Code.
3. Syntax Tree.

Three address code.

Three address code is a type of intermediate code which is easy to generate and can be easily converted to machine code.It makes use of at most three addresses and one operator to represent an expression and the value computed at each instruction is stored in temporary variable generated by compiler. The compiler decides the order of operation given by three address code.
A statement involving no more than three references(two for operands and one for result) is known as three address statement. A sequence of three address statements is known as three address code. Three address statement is of the form x = y op z , here x, y, z will have address (memory location). Sometimes a statement might contain less than three references but it is still called three address statement.
General representation –

$$a = b\ op\ c$$

Where a, b or c represents operands like names, constants or compiler generated temporaries and op represents the operator.
Example – The three address code for the expression a + b * c + d :
$T_1 = b * c$
$T_2 = a + T_1$
$T_3 = T_2 + d$

T_1, T_2, T_3 are temporary variables.

14.3 Algorithm

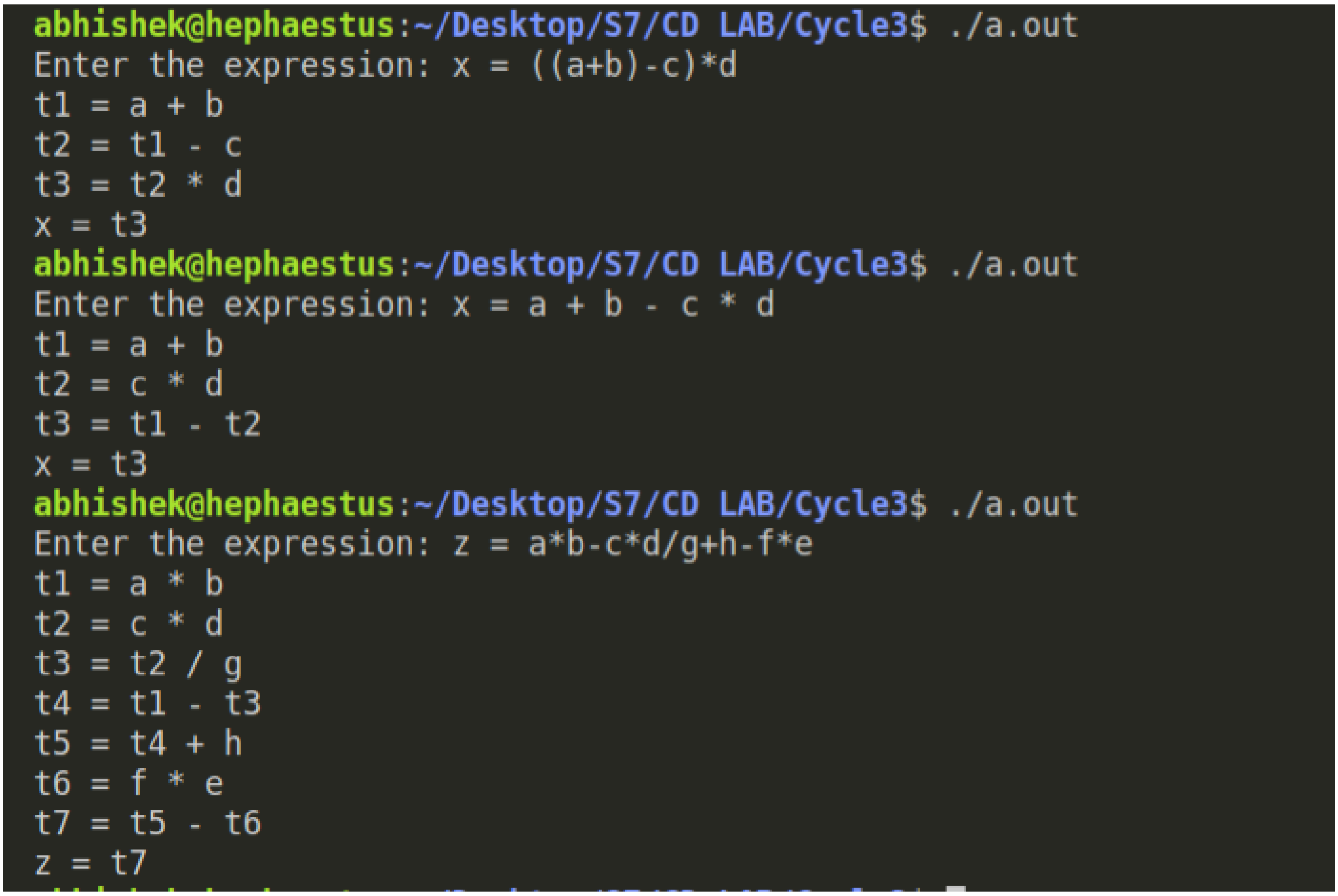
Algorithm 18: Algorithm for 3 address code generation

```
1. while there are still tokens to be read in,
   1.1 Get the next token.
   1.2 if the token is:
       1.2.1 A Variable: push it onto the value stack.
       1.2.2 A left parenthesis: push it onto the operator stack.
       1.2.3 A right parenthesis:
         1 while the thing on top of the operator stack is not a
           left parenthesis,
             1 Pop the operator from the operator stack.
             2 Pop the value stack twice, getting two operands.
             3 Apply the operator to the operands, in the correct order and print.
             4 Push the temporary variable onto the value stack.
         2 Pop the left parenthesis from the operator stack, and discard it.
       1.2.4 An operator (call it thisOp):
         1 while the operator stack is not empty, and the top thing on the
           operator stack has the same or greater precedence as thisOp,
           1 Pop the operator from the operator stack.
           2 Pop the value stack twice, getting two operands.
           3 Apply the operator to the operands, in the correct order and print.
           4 Push the temporary variable onto the value stack.
         2 Push thisOp onto the operator stack.
2. while the operator stack is not empty,
    1 Pop the operator from the operator stack.
    2 Pop the value stack twice, getting two operands.
    3 Apply the operator to the operands, in the correct order and print.
    4 Push the temporary variable onto the value stack.
3. At this point the operator stack should be empty, and the value
   stack should have only one value in it, assign it to the LHS of = variable.
```

14.4 Code

```cpp
#include <bits/stdc++.h>
using namespace std;
int precedence(char a)
{
    if (a == '+' || a == '-')
    {
        return 0;
    }
    if (a == '(')
        return -1;
    return 1;
}
bool isop(char s)
{
    if (s == '+' || s == '-' || s == '*' || s == '/')
    {
        return true;
    }
    else
    {
        return false;
    }
}
void print_star(int n)
{
    for (int i = 0; i < n; ++i)
    {
        cout << "*";
    }
    cout << endl;
}
string charint(char s)
{
    string res = "";
    if (s >= '1' && s <= '9')
    {
        res += 't';
        res += s;
        return res;
    }
    return res + s;
}
int main()
{
    string s;
    cout << "Enter the expression: ";
    getline(cin, s);
    vector<char> input;
    int len = s.size();
    int start = 0;
    while (s[start] != '=')
    {
        start++;
    }
    for (int i = start + 1; i < len; ++i)
    {
        if (s[i] == ' ')
            continue;
        input.push_back(s[i]);
    }
    // for (auto x : input)
    // {
    //      cout << x;
    // }
    char count = '1';
    stack<char> value;
    stack<char> op;
    for (int i = 0; i < input.size(); ++i)
    {
        //cout << input[i] << "current reading" << endl;
        //cout << isop(input[i]);
        if (isalpha(input[i]))
        {
            //cout << input[i] << " pushed into the stack" << endl;
            value.push(input[i]);
        }
        else if (input[i] == '(')
        {
            op.push(input[i]);
        }

        else if (isop(input[i]))
        {
            //cout << "operant found: " << input[i] << endl;
            while (!op.empty() && precedence(op.top()) >= precedence(input[i]))
            {
                char a1, a2, o1;
                a2 = value.top();
                value.pop();
                a1 = value.top();
                value.pop();
                o1 = op.top();
                op.pop();
                string b1, b2;
                b1 = charint(a1);
                b2 = charint(a2);
                cout << "t" << count << " = " << b1 << " " << o1 << " " << b2 << endl;
                value.push(count);
                count++;
            }
            op.push(input[i]);
        }
        else
        { // closing bracket present
            //cout << "closing bracket found " << endl;
            while (!op.empty() && op.top() != '(')
            {
                char a1, a2, o1;
                a2 = value.top();
                value.pop();
                a1 = value.top();
                value.pop();
                o1 = op.top();
                op.pop();
                string b1, b2;
                b1 = charint(a1);
                b2 = charint(a2);
                cout << "t" << count << " = " << b1 << " " << o1 << " " << b2 << endl;
                value.push(count);
                count++;
            }
            op.pop();
        }
    }
    while (!op.empty())
    {
        char a1, a2, o1;
        a2 = value.top();
        value.pop();
        a1 = value.top();
        value.pop();
        o1 = op.top();
        op.pop();
        string b1, b2;
        b1 = charint(a1);
        b2 = charint(a2);
        cout << "t" << count << " = " << b1 << " " << o1 << " " << b2 << endl;
        value.push(count);
        count++;
    }
    cout << s[0] << " = " << charint(count - 1) << endl;
    return 0;
}
```

14.5 Output



```
abhishek@hephaestus:~/Desktop/S7/CD LAB/Cycle3$ ./a.out
Enter the expression: x = ((a+b)-c)*d
t1 = a + b
t2 = t1 - c
t3 = t2 * d
x = t3
abhishek@hephaestus:~/Desktop/S7/CD LAB/Cycle3$ ./a.out
Enter the expression: x = a + b - c * d
t1 = a + b
t2 = c * d
t3 = t1 - t2
x = t3
abhishek@hephaestus:~/Desktop/S7/CD LAB/Cycle3$ ./a.out
Enter the expression: z = a*b-c*d/g+h-f*e
t1 = a * b
t2 = c * d
t3 = t2 / g
t4 = t1 - t3
t5 = t4 + h
t6 = f * e
t7 = t5 - t6
z = t7
```

```
abhishek@hephaestus:~/Desktop/S7/CD LAB/Cycle3$ ./a.out
Enter the expression: x = ((a+b)-c)*d
t1 = a + b
t2 = t1 - c
t3 = t2 * d
x = t3
abhishek@hephaestus:~/Desktop/S7/CD LAB/Cycle3$ ./a.out
```

```
Enter the expression: x = a + b - c * d
t1 = a + b
t2 = c * d
t3 = t1 - t2
x = t3
abhishek@hephaestus:~/Desktop/S7/CD LAB/Cycle3$ ./a.out
Enter the expression: z = a*b-c*d/g+h-f*e
t1 = a * b
t2 = c * d
t3 = t2 / g
t4 = t1 - t3
t5 = t4 + h
t6 = f * e
t7 = t5 - t6
z = t7
abhishek@hephaestus:~/Desktop/S7/CD LAB/Cycle3$
```

14.6 Result

Implemented the program for Intermediate code generation(3 Address code). It was compiled using g++ version 9.3.0, and executed in Ubuntu 20.04 and the above output was obtained.