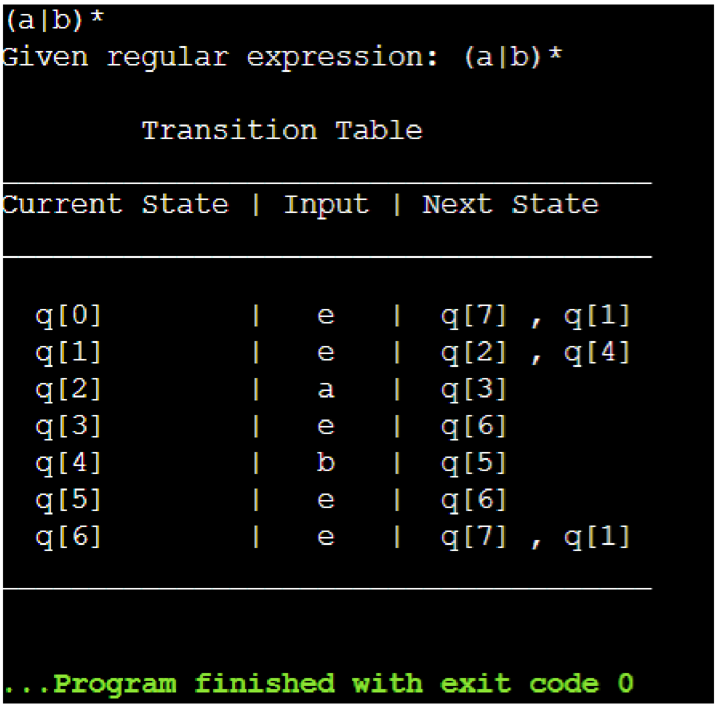
Practical – 5

Objective: WAP to convert regular expression to equivalent NFA**.**

5.A- Source Code:-

```
#include<stdio.h>
#include<string.h>
int main()
{
        char reg[20]; int q[20][3],i=0,j=1,len,a,b;
        for(a=0;a<20;a++) for(b=0;b<3;b++) q[a][b]=0;
        scanf("%s",reg);
        printf("Given regular expression: %s\n",reg);
        len=strlen(reg);
        while(i<len)
        {
                if(reg[i]=='a'&&reg[i+1]!='|'&&reg[i+1]!='*') { q[j][0]=j+1; j++; }
                if(reg[i]=='b'&&reg[i+1]!='|'&&reg[i+1]!='*') {     q[j][1]=j+1; j++;       }
                if(reg[i]=='e'&&reg[i+1]!='|'&&reg[i+1]!='*') {     q[j][2]=j+1; j++;       }
                if(reg[i]=='a'&&reg[i+1]=='|'&&reg[i+2]=='b')
                {
                  q[j][2]=((j+1)*10)+(j+3); j++;
                  q[j][0]=j+1; j++;
                        q[j][2]=j+3; j++;
                        q[j][1]=j+1; j++;
                        q[j][2]=j+1; j++;
                        i=i+2;
                }
                if(reg[i]=='b'&&reg[i+1]=='|'&&reg[i+2]=='a')
                {
                        q[j][2]=((j+1)*10)+(j+3); j++;
                        q[j][1]=j+1; j++;
                        q[j][2]=j+3; j++;
                        q[j][0]=j+1; j++;
                        q[j][2]=j+1; j++;
                        i=i+2;
                }
                if(reg[i]=='a'&&reg[i+1]=='*')
                {
                        q[j][2]=((j+1)*10)+(j+3); j++;
                        q[j][0]=j+1; j++;
                        q[j][2]=((j+1)*10)+(j-1); j++;
                }
                if(reg[i]=='b'&&reg[i+1]=='*')
                {
                        q[j][2]=((j+1)*10)+(j+3); j++;
                        q[j][1]=j+1; j++;
                        q[j][2]=((j+1)*10)+(j-1); j++;
                }
                if(reg[i]==')'&&reg[i+1]=='*')
                {
                        q[0][2]=((j+1)*10)+1;
                        q[j][2]=((j+1)*10)+1;
                        j++;
                }
                i++;
        }
        printf("\n\tTransition Table \n");
        printf("_____________________________________________\n");
        printf("Current State |\tInput |\tNext State");
        printf("\n_____________________________________________\n");
        for(i=0;i<=j;i++)
        {
                if(q[i][0]!=0) printf("\n  q[%d]\t      |  a   |  q[%d]",i,q[i][0]);
                if(q[i][1]!=0) printf("\n  q[%d]\t      |  b   |  q[%d]",i,q[i][1]);
                if(q[i][2]!=0)
                {
                        if(q[i][2]<10) printf("\n  q[%d]\t      |  e   |  q[%d]",i,q[i][2]);
                        else printf("\n  q[%d]\t      |  e   |  q[%d] ,
q[%d]",i,q[i][2]/10,q[i][2]%10);
                }
        }
        printf("\n_____________________________________________\n");
        return 0;
}
```

Output:



```
(a|b)*
Given regular expression: (a|b)*

        Transition Table
_____________________________________________
Current State |  Input |  Next State
_____________________________________________

  q[0]          |   e    |  q[7] , q[1]
  q[1]          |   e    |  q[2] , q[4]
  q[2]          |   a    |  q[3]
  q[3]          |   e    |  q[6]
  q[4]          |   b    |  q[5]
  q[5]          |   e    |  q[6]
  q[6]          |   e    |  q[7] , q[1]
_____________________________________________


...Program finished with exit code 0
```

Practical – 6

Objective: WAP to convert NFA to equivalent DFA.

6.A: Source code:-

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_LEN 100

char NFA_FILE[MAX_LEN];
char buffer[MAX_LEN];
int zz = 0;

struct DFA {
char *states;
int count;
} dfa;

int last_index = 0;
FILE *fp;
int symbols;

void reset(int ar[], int size) {
int i;

for (i = 0; i < size; i++) {
        ar[i] = 0;
}
}

void check(int ar[], char S[]) {
int i, j;

int len = strlen(S);
for (i = 0; i < len; i++) {

        j = ((int)(S[i]) - 65);
        ar[j]++;
}
}

void state(int ar[], int size, char S[]) {
int j, k = 0;
for (j = 0; j < size; j++) {
        if (ar[j] != 0)
        S[k++] = (char)(65 + j);
}
S[k] = '\0';
}
int closure(int ar[], int size) {
int i;
for (i = 0; i < size; i++) {
        if (ar[i] == 1)
        return i;
}
return (100);
}
int indexing(struct DFA *dfa) {
int i;

for (i = 0; i < last_index; i++) {
        if (dfa[i].count == 0)
        return 1;
}
return -1;
}
void Display_closure(int states, int closure_ar[],
                                        char *closure_table[],
                                        char *NFA_TABLE[][symbols + 1],
                                        char *DFA_TABLE[][symbols]) {
int i;
for (i = 0; i < states; i++) {
        reset(closure_ar, states);
        closure_ar[i] = 2;
        if (strcmp(&NFA_TABLE[i][symbols], "-") != 0) {
        strcpy(buffer, &NFA_TABLE[i][symbols]);
        check(closure_ar, buffer);
        int z = closure(closure_ar, states);

        while (z != 100)
        {
                if (strcmp(&NFA_TABLE[z][symbols], "-") != 0) {
                strcpy(buffer, &NFA_TABLE[z][symbols]);

                check(closure_ar, buffer);
                }
                closure_ar[z]++;
                z = closure(closure_ar, states);
        }
        }
        printf("\n e-Closure (%c) :\t", (char)(65 + i));

        bzero((void *)buffer, MAX_LEN);
        state(closure_ar, states, buffer);
        strcpy(&closure_table[i], buffer);
        printf("%s\n", &closure_table[i]);
}
}
```

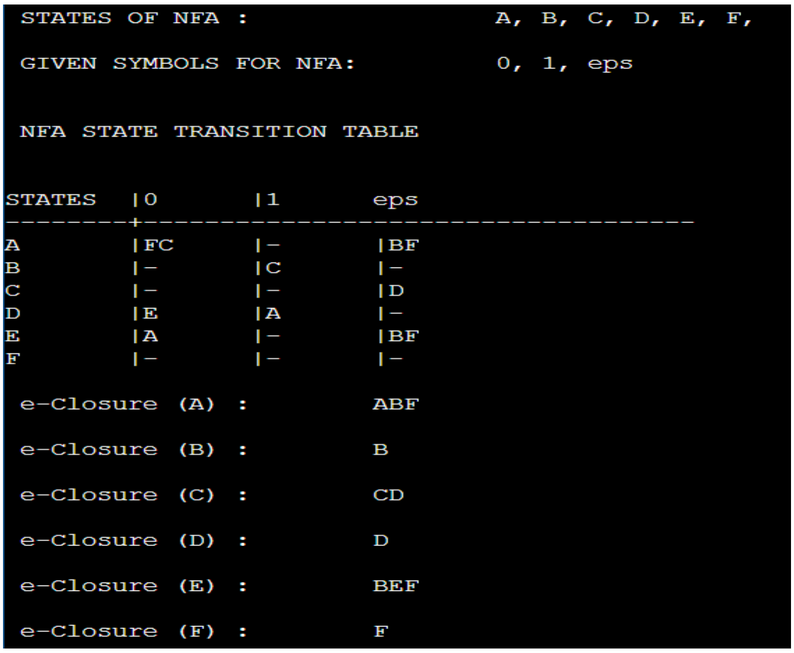
```
int new_states(struct DFA *dfa, char S[]) {

int i;

for (i = 0; i < last_index; i++) {
        if (strcmp(&dfa[i].states, S) == 0)
        return 0;
}

strcpy(&dfa[last_index++].states, S);

dfa[last_index - 1].count = 0;
return 1;
}

void trans(char S[], int M, char *clsr_t[], int st,
                        char *NFT[][symbols + 1], char TB[]) {
int len = strlen(S);
int i, j, k, g;
int arr[st];
int sz;
reset(arr, st);
char temp[MAX_LEN], temp2[MAX_LEN];
char *buff;

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

        j = ((int)(S[i] - 65));
        strcpy(temp, &NFT[j][M]);

        if (strcmp(temp, "-") != 0) {
        sz = strlen(temp);
        g = 0;

        while (g < sz) {
                k = ((int)(temp[g] - 65));
                strcpy(temp2, &clsr_t[k]);
                check(arr, temp2);
                g++;
        }
        }
}

bzero((void *)temp, MAX_LEN);
state(arr, st, temp);
if (temp[0] != '\0') {
        strcpy(TB, temp);
} else
        strcpy(TB, "-");
}
```

```
/* Display DFA transition state table*/
void Display_DFA(int last_index, struct DFA *dfa_states,
                char *DFA_TABLE[][symbols]) {
int i, j;
printf("\n\n****************************************************************\n\n");
printf("\t\t DFA TRANSITION STATE TABLE \t\t \n\n");
printf("\n STATES OF DFA :\t\t");

for (i = 1; i < last_index; i++)
    printf("%s, ", &dfa_states[i].states);
printf("\n");
printf("\n GIVEN SYMBOLS FOR DFA: \t");

for (i = 0; i < symbols; i++)
    printf("%d, ", i);
printf("\n\n");
printf("STATES\t");

for (i = 0; i < symbols; i++)
    printf("|%d\t", i);
printf("\n");

// display the DFA transition state table
printf("--------+-----------------------\n");
for (i = 0; i < zz; i++) {
    printf("%s\t", &dfa_states[i + 1].states);
    for (j = 0; j < symbols; j++) {
    printf("|%s \t", &DFA_TABLE[i][j]);
    }
    printf("\n");
}
}

// Driver Code
int main() {
int i, j, states;
char T_buf[MAX_LEN];

struct DFA *dfa_states = malloc(MAX_LEN * (sizeof(dfa)));
states = 6, symbols = 2;

printf("\n STATES OF NFA :\t\t");
for (i = 0; i < states; i++)

    printf("%c, ", (char)(65 + i));
printf("\n");
printf("\n GIVEN SYMBOLS FOR NFA: \t");

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

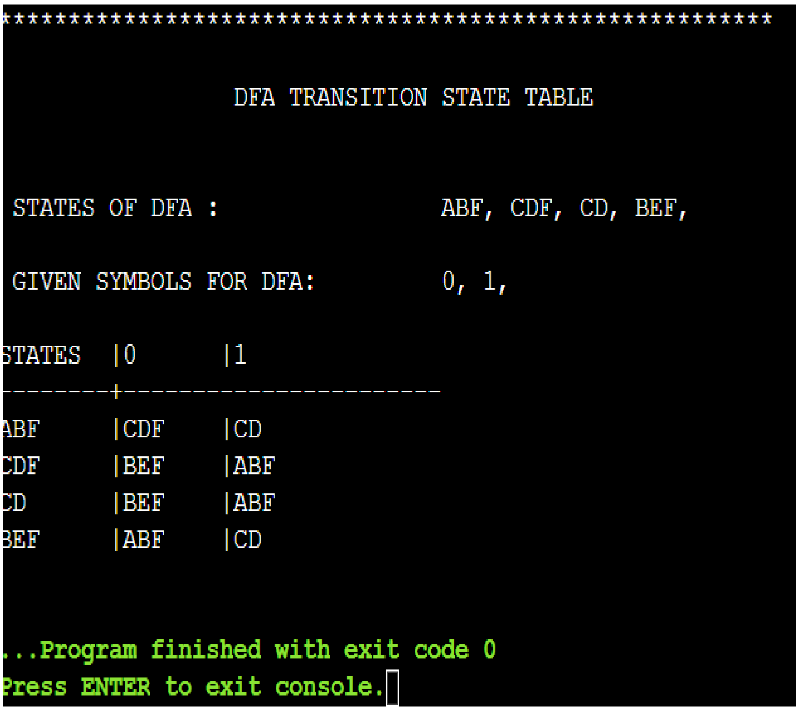
```
        printf("%d, ", i);
printf("eps");
printf("\n\n");
char *NFA_TABLE[states][symbols + 1];

char *DFA_TABLE[MAX_LEN][symbols];
strcpy(&NFA_TABLE[0][0], "FC");
strcpy(&NFA_TABLE[0][1], "-");
strcpy(&NFA_TABLE[0][2], "BF");
strcpy(&NFA_TABLE[1][0], "-");
strcpy(&NFA_TABLE[1][1], "C");
strcpy(&NFA_TABLE[1][2], "-");
strcpy(&NFA_TABLE[2][0], "-");
strcpy(&NFA_TABLE[2][1], "-");
strcpy(&NFA_TABLE[2][2], "D");
strcpy(&NFA_TABLE[3][0], "E");
strcpy(&NFA_TABLE[3][1], "A");
strcpy(&NFA_TABLE[3][2], "-");
strcpy(&NFA_TABLE[4][0], "A");
strcpy(&NFA_TABLE[4][1], "-");
strcpy(&NFA_TABLE[4][2], "BF");
strcpy(&NFA_TABLE[5][0], "-");
strcpy(&NFA_TABLE[5][1], "-");
strcpy(&NFA_TABLE[5][2], "-");
printf("\n NFA STATE TRANSITION TABLE \n\n\n");
printf("STATES\t");

for (i = 0; i < symbols; i++)
        printf("|%d\t", i);
printf("eps\n");

// Displaying the matrix of NFA transition table
printf("--------+------------------------------------\n");
for (i = 0; i < states; i++) {
        printf("%c\t", (char)(65 + i));

        for (j = 0; j <= symbols; j++) {
        printf("|%s \t", &NFA_TABLE[i][j]);
        }
        printf("\n");
}
int closure_ar[states];
char *closure_table[states];

Display_closure(states, closure_ar, closure_table, NFA_TABLE, DFA_TABLE);
strcpy(&dfa_states[last_index++].states, "-");

dfa_states[last_index - 1].count = 1;
bzero((void *)buffer, MAX_LEN);
```

```
strcpy(buffer, &closure_table[0]);
strcpy(&dfa_states[last_index++].states, buffer);

int Sm = 1, ind = 1;
int start_index = 1;

while (ind != -1) {
        dfa_states[start_index].count = 1;
        Sm = 0;
        for (i = 0; i < symbols; i++) {

        trans(buffer, i, closure_table, states, NFA_TABLE, T_buf);

        strcpy(&DFA_TABLE[zz][i], T_buf);

        Sm = Sm + new_states(dfa_states, T_buf);
        }
        ind = indexing(dfa_states);
        if (ind != -1)
        strcpy(buffer, &dfa_states[++start_index].states);
        zz++;
}

Display_DFA(last_index, dfa_states, DFA_TABLE);

return 0;
}
```

Output:



```
 STATES OF NFA :                A, B, C, D, E, F,

 GIVEN SYMBOLS FOR NFA:         0, 1, eps


 NFA STATE TRANSITION TABLE


STATES   |0       |1       eps
--------+------------------------------------
A        |FC      |-       |BF
B        |-       |C       |-
C        |-       |-       |D
D        |E       |A       |-
E        |A       |-       |BF
F        |-       |-       |-

 e-Closure (A) :          ABF

 e-Closure (B) :          B

 e-Closure (C) :          CD

 e-Closure (D) :          D

 e-Closure (E) :          BEF

 e-Closure (F) :          F
```



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


                 DFA TRANSITION STATE TABLE



 STATES OF DFA :                ABF, CDF, CD, BEF,


 GIVEN SYMBOLS FOR DFA:         0, 1,


STATES   |0       |1
--------+-----------------------
ABF      |CDF     |CD
CDF      |BEF     |ABF
CD       |BEF     |ABF
BEF      |ABF     |CD



...Program finished with exit code 0
Press ENTER to exit console.
```

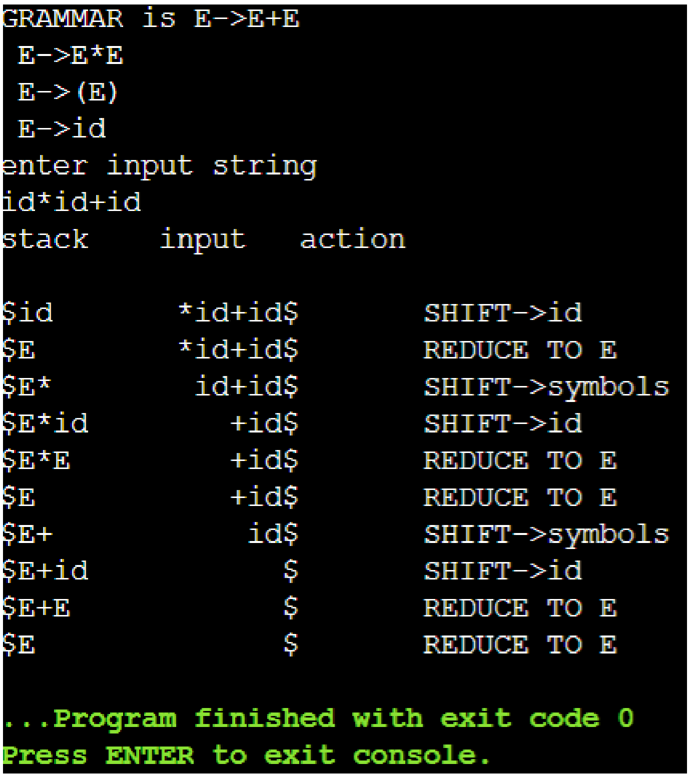
Practical – 7

Objective: WAP to implement shift reduce parser.

7.A- Source Code:-

```
#include<stdio.h>
#include<string.h>
int k=0,z=0,i=0,j=0,c=0;
char a[16],ac[20],stk[15],act[10];
void check();
int main()
  {

     puts("GRAMMAR is E->E+E \n E->E*E \n E->(E) \n E->id");
     puts("enter input string ");
     gets(a);
     c=strlen(a);
     strcpy(act,"SHIFT->");
     puts("stack \t input \t action");
     for(k=0,i=0; j<c; k++,i++,j++){
       if(a[j]=='i' && a[j+1]=='d'){
            stk[i]=a[j];
            stk[i+1]=a[j+1];
            stk[i+2]='\0';
            a[j]=' ';
            a[j+1]=' ';
            printf("\n$%s\t%s$\t%sid",stk,a,act);
            check();
          }
       else{
            stk[i]=a[j];
            stk[i+1]='\0';
            a[j]=' ';
            printf("\n$%s\t%s$\t%ssymbols",stk,a,act);
            check();
          }
      }

  }
void check(){
    strcpy(ac,"REDUCE TO E");
    for(z=0; z<c; z++)
      if(stk[z]=='i' && stk[z+1]=='d'){
          stk[z]='E';
          stk[z+1]='\0';
          printf("\n$%s\t%s$\t%s",stk,a,ac);
          j++;
      }
    for(z=0; z<c; z++)
     if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='E')
      {
       stk[z]='E';
       stk[z+1]='\0';
       stk[z+2]='\0';
       printf("\n$%s\t%s$\t%s",stk,a,ac);
       i=i-2;
      }
    for(z=0; z<c; z++){
     if(stk[z]=='E' && stk[z+1]=='*' && stk[z+2]=='E') {
       stk[z]='E';
       stk[z+1]='\0';
       stk[z+1]='\0';
       printf("\n$%s\t%s$\t%s",stk,a,ac);
       i=i-2;
      }
    for(z=0; z<c; z++){
     if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')') {
       stk[z]='E';
       stk[z+1]='\0';
       stk[z+1]='\0';
       printf("\n$%s\t%s$\t%s",stk,a,ac);
       i=i-2;
      }
   }
```

Output:



```
GRAMMAR is E->E+E
 E->E*E
 E->(E)
 E->id
enter input string
id*id+id
stack    input    action

$id       *id+id$       SHIFT->id
$E        *id+id$       REDUCE TO E
$E*        id+id$       SHIFT->symbols
$E*id        +id$       SHIFT->id
$E*E         +id$       REDUCE TO E
$E           +id$       REDUCE TO E
$E+           id$       SHIFT->symbols
$E+id           $       SHIFT->id
$E+E            $       REDUCE TO E
$E              $       REDUCE TO E

...Program finished with exit code 0
Press ENTER to exit console.
```

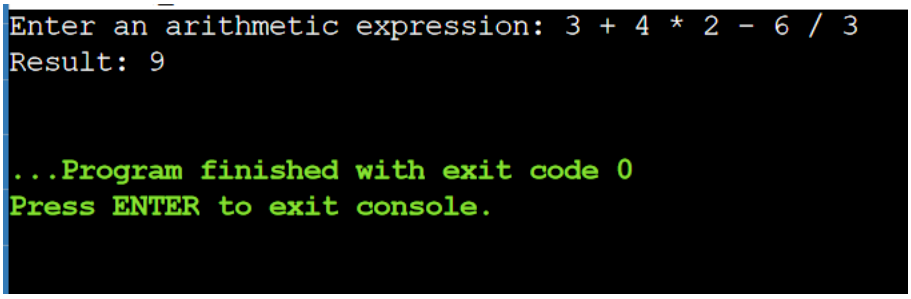
Practical – 8

Objective: WAP to implement Operator Precedence parser.

8.A- Source Code:-

```
#include <iostream>
#include <stack>
#include <string>

using namespace std;

// Function to check if a character is an operator
bool isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/');
}

// Function to get the precedence of an operator
int getPrecedence(char op) {
    if (op == '+' || op == '-')
        return 1;
    else if (op == '*' || op == '/')
        return 2;
    return 0;
}

// Function to perform an operation
int performOperation(int operand1, int operand2, char op) {
    switch (op) {
        case '+':
            return operand1 + operand2;
        case '-':
            return operand1 - operand2;
        case '*':
            return operand1 * operand2;
        case '/':
            if (operand2 != 0)
                return operand1 / operand2;
            else {
                cout << "Error: Division by zero" << endl;
                exit(1);
            }
    }
    return 0;
}

// Function to evaluate the expression using operator precedence parsing
int evaluateExpression(const string& expression) {
    stack<int> operandStack;
    stack<char> operatorStack;

    for (char c : expression) {
        if (isspace(c)) {
            continue;
        } else if (isdigit(c)) {
            int operand = c - '0';
            operandStack.push(operand);
        } else if (isOperator(c)) {
            while (!operatorStack.empty() && getPrecedence(operatorStack.top()) >=
getPrecedence(c)) {
                int operand2 = operandStack.top();
                operandStack.pop();

                int operand1 = operandStack.top();
                operandStack.pop();

                char op = operatorStack.top();
                operatorStack.pop();

                int result = performOperation(operand1, operand2, op);
                operandStack.push(result);
            }

            operatorStack.push(c);
        } else {
            cout << "Error: Invalid character '" << c << "'" << endl;
            exit(1);
        }
    }

    while (!operatorStack.empty()) {
        int operand2 = operandStack.top();
        operandStack.pop();

        int operand1 = operandStack.top();
        operandStack.pop();
        char op = operatorStack.top();
        operatorStack.pop();

        int result = performOperation(operand1, operand2, op);
        operandStack.push(result);
    }

    return operandStack.top();
}

int main() {
    string expression;
    cout << "Enter an arithmetic expression: ";
    getline(cin, expression);

    int result = evaluateExpression(expression);
    cout << "Result: " << result << endl;

    return 0;
}
```

Output:



```
Enter an arithmetic expression: 3 + 4 * 2 - 6 / 3
Result: 9


...Program finished with exit code 0
Press ENTER to exit console.
```

Practical – 9

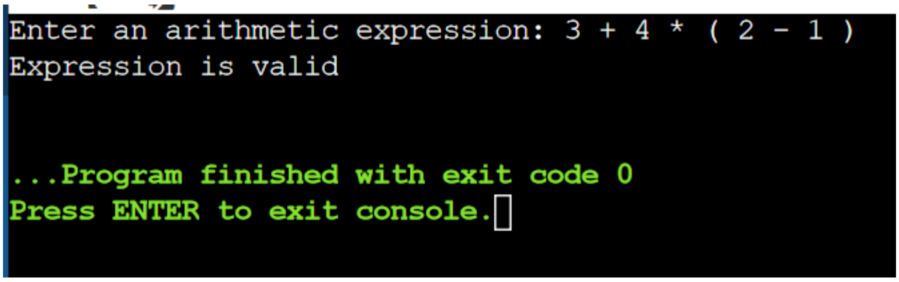
Objective: WAP to implement Recursive Descent parser.

9.A- Source Code:-

```
#include <iostream>
#include <cctype>
#include <cstdlib>
#include <algorithm> // Include the <algorithm> header for remove_if

using namespace std;

string input;
size_t position = 0;

void error() {
   cout << "Error: Invalid expression" << endl;
   exit(1);
}

char getNextToken() {
   return input[position++];
}

void factor();
void term();
void expr();

void factor() {
   char token = getNextToken();
   if (isdigit(token)) {
      // Valid factor
   } else if (token == '(') {
      expr();
      token = getNextToken();
      if (token != ')')
         error();
   } else {
      error();
   }
}
```

```
void term() {
    factor();
    char token = getNextToken();
    while (token == '*' || token == '/') {
        factor();
        token = getNextToken();
    }
    position--; // Move the position back to the last valid token
}

void expr() {
    term();
    char token = getNextToken();
    while (token == '+' || token == '-') {
        term();
        token = getNextToken();
    }
    position--; // Move the position back to the last valid token
}

int main() {
    cout << "Enter an arithmetic expression: ";
    getline(cin, input);
    input.erase(remove_if(input.begin(), input.end(), [](char c) { return isspace(c); }),
input.end());
    input += '$'; // Add end marker
    expr();
    if (input[position] == '$') {
        cout << "Expression is valid" << endl;
    } else {
        cout << "Expression is invalid" << endl;
    }
    return 0;
}
```

Output:



```
Enter an arithmetic expression: 3 + 4 * ( 2 - 1 )
Expression is valid


...Program finished with exit code 0
Press ENTER to exit console.
```

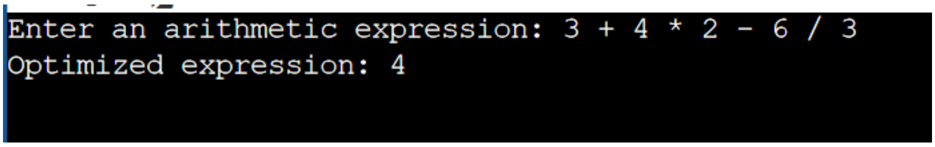
Practical – 10

Objective: WAP to implement Code Optimization Techniques.

10.A- Source Code:-

```
#include <iostream>
#include <string>
#include <cctype>

using namespace std;

int evaluateExpression(int operand1, int operand2, char op) {
    switch (op) {
        case '+':
            return operand1 + operand2;
        case '-':
            return operand1 - operand2;
        case '*':
            return operand1 * operand2;
        case '/':
            if (operand2 != 0)
                return operand1 / operand2;
            else {
                cout << "Error: Division by zero" << endl;
                exit(1);
            }
    }
    return 0;
}

string optimizeExpression(const string& expression) {
    int operand1 = 0;
    int operand2 = 0;
    char op = '+';
    int result = 0;

    for (char c : expression) {
        if (isspace(c)) {
            continue;
        } else if (isdigit(c)) {
            operand2 = operand2 * 10 + (c - '0');
        } else {
            result = evaluateExpression(result, operand2, op);
            op = c;
            operand2 = 0;
        }
    }

    result = evaluateExpression(result, operand2, op);
    return to_string(result);
}

int main() {
    string expression;
    cout << "Enter an arithmetic expression: ";
    getline(cin, expression);

    string optimizedExpression = optimizeExpression(expression);
    cout << "Optimized expression: " << optimizedExpression << endl;

    return 0;
}
```

Output:



```
Enter an arithmetic expression: 3 + 4 * 2 - 6 / 3
Optimized expression: 4
```

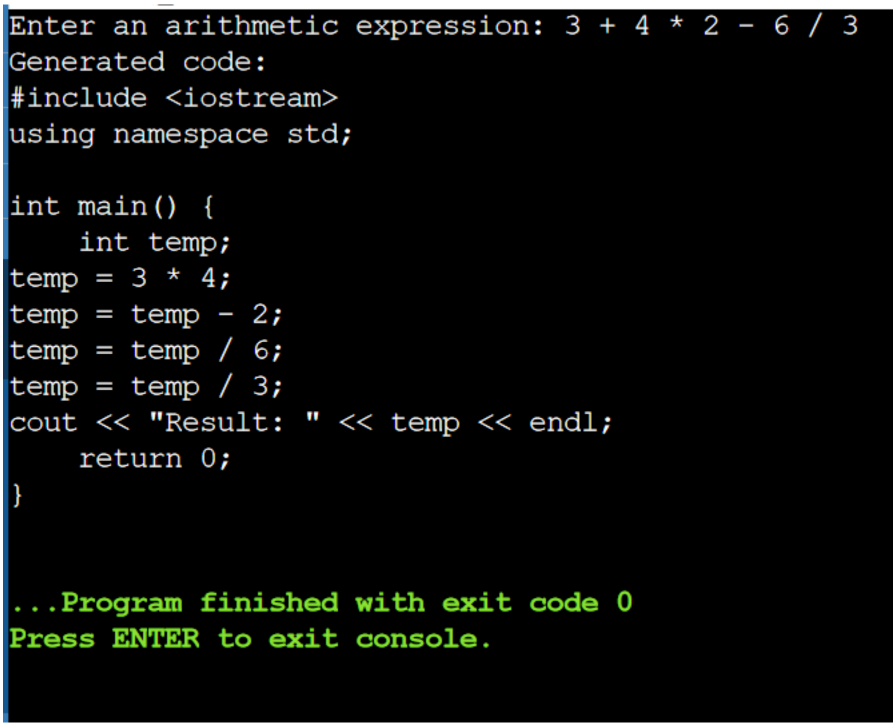
Practical – 11

Objective: WAP to implement Code Generator.

11.A- Source Code:-

```
#include <iostream>
#include <string>
#include <stack>
#include <cctype>

using namespace std;

string generateCode(const string& expression) {
    string code;
    stack<string> operandStack;
    stack<char> operatorStack;

    for (size_t i = 0; i < expression.length(); i++) {
        char c = expression[i];

        if (isspace(c)) {
            continue;
        } else if (isdigit(c)) {
            size_t j = i;
            string number;
            while (j < expression.length() && isdigit(expression[j])) {
                number += expression[j];
                j++;
            }
            operandStack.push(number);
            i = j - 1;
        } else if (c == '+' || c == '-' || c == '*' || c == '/') {
            while (!operatorStack.empty() && operatorStack.top() != '(') {
                string operand2 = operandStack.top();
                operandStack.pop();
                string operand1 = operandStack.top();
                operandStack.pop();

                code += "temp = " + operand1 + " " + c + " " + operand2 + ";\n";
                operandStack.push("temp");
                operatorStack.pop();
            }
            operatorStack.push(c);
        } else if (c == '(') {
            operatorStack.push(c);
        } else if (c == ')') {
            while (!operatorStack.empty() && operatorStack.top() != '(') {
                string operand2 = operandStack.top();
                operandStack.pop();
                string operand1 = operandStack.top();
                operandStack.pop();

                code += "temp = " + operand1 + " " + operatorStack.top() + " " + operand2 + ";\n";
                operandStack.push("temp");
                operatorStack.pop();
            }

            if (!operatorStack.empty())
                operatorStack.pop(); // Remove '('
        } else {
            cout << "Error: Invalid character '" << c << "'" << endl;
            exit(1);
        }
    }

    while (!operatorStack.empty()) {
        string operand2 = operandStack.top();
        operandStack.pop();
        string operand1 = operandStack.top();
        operandStack.pop();

        code += "temp = " + operand1 + " " + operatorStack.top() + " " + operand2 + ";\n";
        operandStack.push("temp");
        operatorStack.pop();
    }

    code += "cout << \"Result: \" << temp << endl;\n";
    return code;
}

int main() {
    string expression;
    cout << "Enter an arithmetic expression: ";
    getline(cin, expression);

    string code = generateCode(expression);

    cout << "Generated code:\n";
    cout << "#include <iostream>\n";
    cout << "using namespace std;\n\n";
    cout << "int main() {\n";
    cout << "    int temp;\n";
    cout << code;
    cout << "    return 0;\n";
    cout << "}\n";

    return 0;
}
```

Output:



```
Enter an arithmetic expression: 3 + 4 * 2 - 6 / 3
Generated code:
#include <iostream>
using namespace std;

int main() {
    int temp;
temp = 3 * 4;
temp = temp - 2;
temp = temp / 6;
temp = temp / 3;
cout << "Result: " << temp << endl;
    return 0;
}


...Program finished with exit code 0
Press ENTER to exit console.
```

Graphic Era
HILL UNIVERSITY
Established by an Act of the State Legislature of Uttarakhand (Adhiniyam Sankhya 12 of 2011)
University under section 2(f) of UGC Act, 1956

Computer Science and Engineering Department
Compiler Design Lab (PCS-601)

Index/List Of Experiments

Sr. No.	Experiment	Date	Signature