Compiler Design Lab Assignment
CSB 353

Submitted to-
Dr Anurag Singh

Submitted by-
Aniket(171210008)

LAB-07

Q.1) Write a program to check whether the given grammar is LL1 or not and construct the table.

Source Code:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
char s[20], stack[20];
void main()
{
   char m[5][6][3] = { "tb", " ", " ", "tb", " ", " ", " ", " ", "+tb", " ", " ", "n", "n", "fc", " ", " ", " ", "fc", " ", " ", " ", " ",
"n", "*fc", " a ", "n", "n", "i", " ", " ", " ", "(e)", " ", " ", " " };

   int size[5][6] = { 2, 0, 0, 2, 0, 0, 0, 3, 0, 0, 1, 1, 2, 0, 0, 2, 0, 0, 0, 1, 3, 0, 1, 1, 1, 0, 0, 3, 0, 0 };
   int i, j, k, n, str1, str2;
   clrscr();
   printf("\n Enter the input string: ");
   scanf("%s", s);
   strcat(s, "$");
   n = strlen(s);
   stack[0] = '$';
   stack[1] = 'e';
   i = 1;
   j = 0;
   printf("\nStack     Input\n");
   printf("_____\n");
   while ((stack[i] != '$') && (s[j] != '$'))
   {
      if (stack[i] == s[j])
      {
         i--;
         j++;
      }
      switch (stack[i])
      {
      case 'e':
         str1 = 0;
         break;
```

```
case 'b':
    str1 = 1;
    break;
case 't':
    str1 = 2;
    break;
case 'c':
    str1 = 3;
    break;
case 'f':
    str1 = 4;
    break;
}
switch (s[j])
{
case 'i':
    str2 = 0;
    break;
case '+':
    str2 = 1;
    break;
case '*':
    str2 = 2;
    break;
case '(':
    str2 = 3;
    break;
case ')':
    str2 = 4;
    break;
case '$':
    str2 = 5;
    break;
}
if (m[str1][str2][0] == '\0')
{
    printf("\nERROR");
    exit(0);
}
else if (m[str1][str2][0] == 'n')
    i--;
else if (m[str1][str2][0] == 'i')
    stack[i] = 'i';
```

```
        else
        {
            for (k = size[str1][str2] - 1; k >= 0; k--)          {              stack[i] = m[str1][str2][k];
                i++;
            }
            i--;
        }
        for (k = 0; k <= i; k++)
            printf(" %c", stack[k]);
        printf("       ");
        for (k = j; k <= n; k++)
            printf("%c", s[k]);
        printf(" \n ");
    }
    printf("\n SUCCESS");
    getch();
}
```

```
 Enter the input string: i*i+i

Stack     Input
_____________________
 $ b t       i*i+i$
  $ b c f       i*i+i$
  $ b c i       i*i+i$
  $ b c f *       *i+i$
  $ b c i       i+i$
  $ b       +i$
  $ b t +       +i$
  $ b c f       i$
  $ b c i       i$
  $ b       $

 SUCCESS
Process returned 9 (0x9)   execution time : 17.841 s
Press any key to continue.
```

Lab 8

- Write a program to construct LR0 and SLR1 tables for the given grammar.

Source Code:

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

/* SLR parser for the grammar
   E->E+T (1)
   E->T (2)
   T->T*F (3)
   T->F (4)
   F->(E) (5)
   F->ID (6)
*/

/* Stack structure and fuctions */

/* The only thing important here is that the value associated
with each name of the macro should be unique. They have no realtion to
the states in the dfa */

#define S4 1
#define S5 2
#define S6 3
#define S7 4
#define R1 5
#define R2 6
#define R3 7
#define R4 8
#define R5 9
#define R6 10
#define S11 11
#define AC 11           /* ACCEPT */
#define ER -1           /* ERROR */

/* the parsing table  */
int table[][9]= {
```

```
  {S5,ER,ER,S4,ER,ER,1,2,3},
  {ER,S6,ER,ER,ER,AC,ER,ER,ER},
  {ER,R2,S7,ER,R2,R2,ER,ER,ER},
  {ER,R4,R4,ER,R4,R4,ER,ER,ER},
  {S5,ER,ER,S4,ER,ER,8,2,3},
  {ER,R6,R6,ER,R6,R6,ER,ER,ER},
  {S5,ER,ER,S4,ER,ER,ER,9,3},
  {S5,ER,ER,S4,ER,ER,ER,ER,10},
  {ER,S6,ER,ER,S11,ER,ER,ER,ER},
  {ER,R1,S7,ER,R1,R1,ER,ER,ER},
  {ER,R3,R3,ER,R3,R3,ER,ER,ER},
  {ER,R5,R5,ER,R5,R5,ER,ER,ER}
};
#define STRING_SIZE 20

char string[STRING_SIZE];
int i=0;
int save;

#define STACK_SIZE 40

typedef struct {
  int list[STACK_SIZE];
  int top;
}Stack;

void initialize(Stack *s) {
  s->top=-1;
}
void push(int value,Stack *s) {
  s->list[++(s->top)]=value;
}
int pop(Stack *s) {
  return(s->list[(s->top)--]);
}
int isempty(Stack *s) {
  return(s->top==-1);
}
int peek(Stack *s) {
  return(s->list[s->top]);
}
int stacksize(Stack *s) {
  return((s->top)+1);
```

```
}

#define ID 0
#define ADD 1
#define MULT 2
#define OPBR 3
#define CLBR 4
#define DOLLAR 5
#define E 6
#define T 7
#define F 8

short int gettoken() {
  /* ignore blanks */
  while(string[i]==' ')
    i++;
  /* definition for identifier */
  if(isalpha(string[i])) {
    save=i;
    i++;
    while(string[i]!=0 && string[i]!='*' && string[i]!='+' && string[i]!=')' && string[i]!='(') {
      if(!isalnum(string[i++]))
    error();
    }
    return ID;
  }
  else if(string[i]=='+') {
    save=i;
    i++;
    return ADD;
  }
  else if(string[i]=='*') {
    save=i;
    i++;
    return MULT;
  }
  else if(string[i]=='(') {
    save=i;
    i++;
    return OPBR;
  }
  else if(string[i]==')') {
    save=i;
```

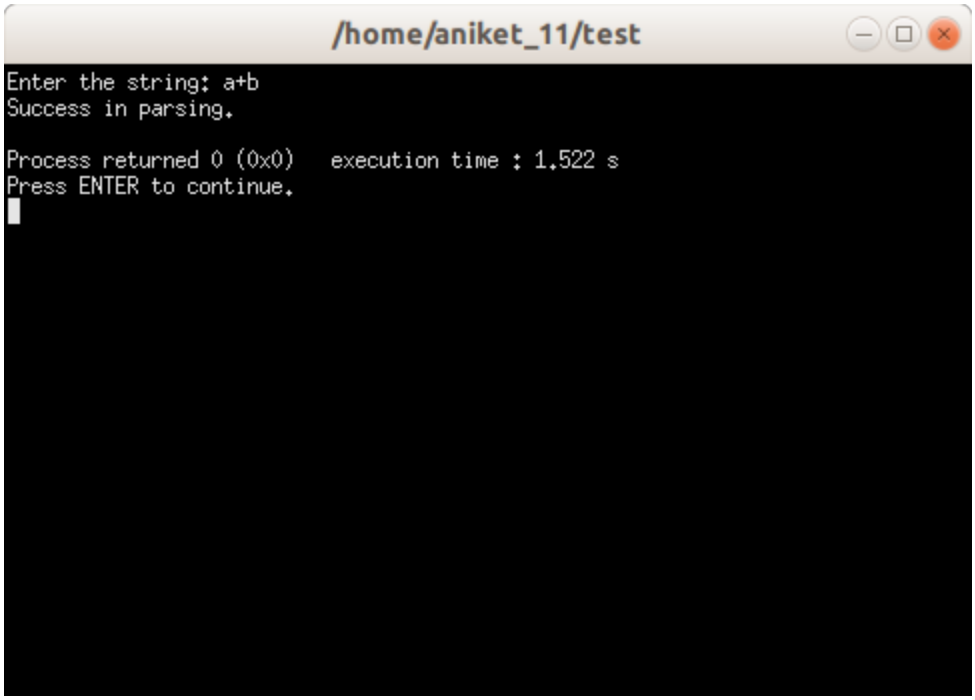
```
    i++;
    return CLBR;
  }
  else if(string[i]==0) {
    return DOLLAR;
  }
}
error() {
  printf("Bad Bad error.\n");
  exit(0);
}
int parse() {
  int token;
  Stack stack;
  int state;
  int j;
  int action;
  int previous;
  initialize(&stack);
  push(0,&stack);
  token=0;
  while(1) {
    token=gettoken();
    action=table[peek(&stack)][token];
    switch(action) {
    case S4:
      push(token,&stack);
      push(4,&stack);
      break;
    case S5:
      push(token,&stack);
      push(5,&stack);
      break;
    case S6:
      push(token,&stack);
      push(6,&stack);
      break;
    case S7:
      push(token,&stack);
      push(7,&stack);
      break;
    case AC:
      return 1;
```

```
case ER:
  error();

}
if(action>=5 && action <=10) {
  while(action>=5 && action <=10) {
action=table[peek(&stack)][token];
switch(action) {
case R1:
  for(j=0;j<6;j++)
    pop(&stack);
  state=table[peek(&stack)][E];
  if(state!=ER) {
    push(E,&stack);
    push(state,&stack);
  }
  else
    error();
  break;
case R2:
  pop(&stack);
  pop(&stack);
  state=table[peek(&stack)][E];
  if(state!=ER) {
    push(E,&stack);
    push(state,&stack);
  }
  else
    error();
  break;
case R3:
  for(j=0;j<6;j++)
    pop(&stack);
  state=table[peek(&stack)][T];
  if(state!=ER) {
    push(T,&stack);
    push(state,&stack);
  }
  else
    error();
  break;
case R4:
  pop(&stack);
```

```
      pop(&stack);
      state=table[peek(&stack)][T];
      if(state!=ER) {
        push(T,&stack);
        push(state,&stack);
      }
      else
        error();
      break;
    case R5:
      for(j=0;j<6;j++)
        pop(&stack);
      state=table[peek(&stack)][F];
      if(state!=ER) {
        push(F,&stack);
        push(state,&stack);
      }
      else
        error();
      break;
    case R6:
      pop(&stack);
      pop(&stack);
      state=table[peek(&stack)][F];
      if(state!=ER) {
        push(F,&stack);
        push(state,&stack);
      }
      else
        error();
      break;
    case AC:
      return 1;
    case ER:
      error();
    }
      }
      i=save;
    }
  }
  return 0;
}
int main() {
```

```
  printf("Enter the string: ");
  scanf("%s",string);
  if(parse()) {
    printf("Success in parsing.\n");
  }
  else
    error();
}
```

Output



LAB-9

Q.1) Write a C program to implement LALR Parser or Lookahead-LR parser.

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void push(char*, int*, char);
char stacktop(char*);
void isproduct(char, char);
int ister(char);
int isnter(char);
int isstate(char);
void error();
void isreduce(char, char);
char pop(char*, int*);
void printt(char*, int*, char[], int);
void rep(char[], int);
struct action {
  char row[6][5];
};

const struct action A[12] = {

    {"sf", "emp", "emp", "se", "emp", "emp"},
    {"emp", "sg", "emp", "emp", "emp", "acc"},
    {"emp", "rc", "sh", "emp", "rc", "rc"},
    {"emp", "re", "re", "emp", "re", "re"},
    {"sf", "emp", "emp", "se", "emp", "emp"},
    {"emp", "rg", "rg", "emp", "rg", "rg"},
    {"sf", "emp", "emp", "se", "emp", "emp"},
    {"sf", "emp", "emp", "se", "emp", "emp"},
    {"emp", "sg", "emp", "emp", "sl", "emp"},
    {"emp", "rb", "sh", "emp", "rb", "rb"},
    {"emp", "rb", "rd", "emp", "rd", "rd"},
    {"emp", "rf", "rf", "emp", "rf", "rf"}};
struct gotol {
  char r[3][4];
};
const struct gotol G[12] = {

    {"b", "c", "d"},      {"emp", "emp", "emp"}, {"emp", "emp", "emp"},
    {"emp", "emp", "emp"}, {"i", "c", "d"},      {"emp", "emp", "emp"},
    {"emp", "j", "d"},    {"emp", "emp", "k"},   {"emp", "emp", "emp"},
```

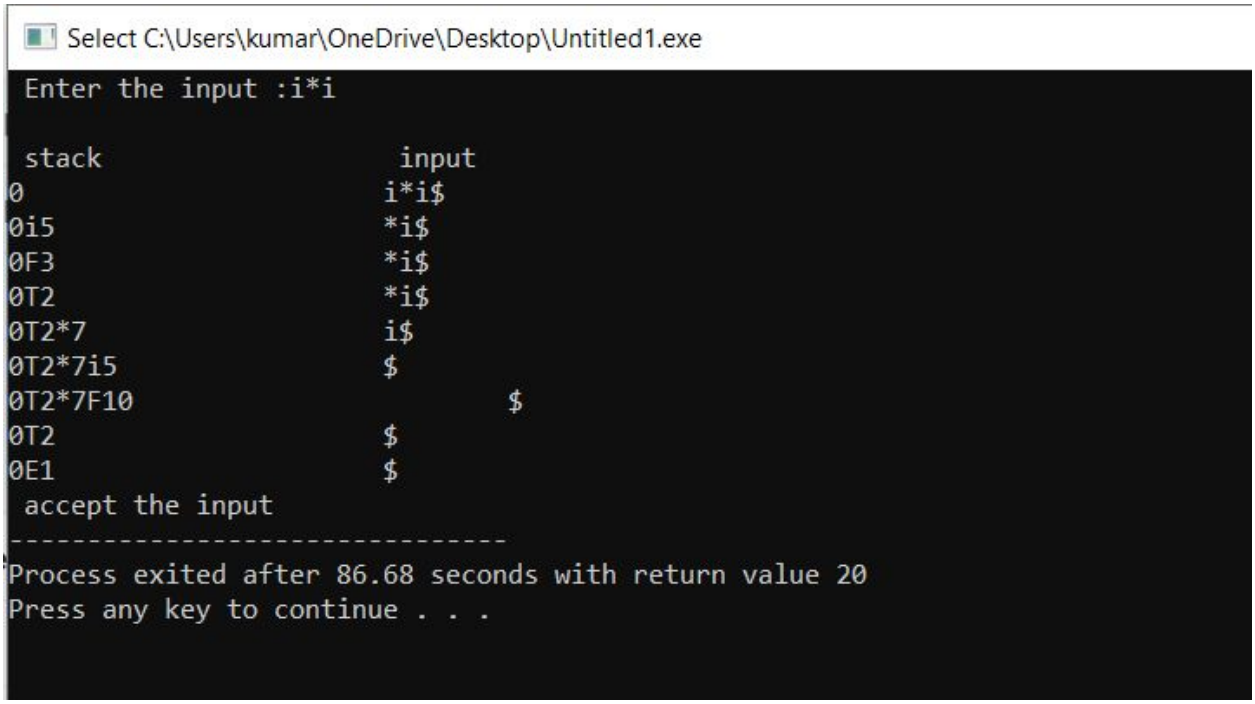
```
    {"emp", "emp", "emp"},
};
char ter[6] = {'i', '+', '*', ')', '(', '$'};
char nter[3] = {'E', 'T', 'F'};
char states[12] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'm', 'j', 'k', 'l'};
char stack[100];
int top = -1;
char temp[10];
struct grammar {
  char left;
  char right[5];
};
const struct grammar rl[6] = {
    {'E', "e+T"}, {'E', "T"},   {'T', "T*F"},
    {'T', "F"},   {'F', "(E)"}, {'F', "i"},
};

void main() {
  char inp[80], x, p, dl[80], y, bl = 'a';
  int i = 0, j, k, l, n, m, c, len;
  printf(" Enter the input :");
  scanf("%s", inp);
  len = strlen(inp);
  inp[len] = '$';
  inp[len + 1] = '\0';
  push(stack, &top, bl);
  printf("\n stack \t\t\t input");
  printt(stack, &top, inp, i);
  do {
    x = inp[i];
    p = stacktop(stack);
    isproduct(x, p);
    if (strcmp(temp, "emp") == 0) error();
    if (strcmp(temp, "acc") == 0)
      break;
    else {
      if (temp[0] == 's') {
        push(stack, &top, inp[i]);
        push(stack, &top, temp[1]);
        i++;
      } else {
        if (temp[0] == 'r') {
          j = isstate(temp[1]);
          strcpy(temp, rl[j - 2].right);
          dl[0] = rl[j - 2].left;
          dl[1] = '\0';
          n = strlen(temp);
          for (k = 0; k < 2 * n; k++) pop(stack, &top);
```

```
          for (m = 0; dl[m] != '\0'; m++) push(stack, &top, dl[m]);
          l = top;
          y = stack[l - 1];
          isreduce(y, dl[0]);
          for (m = 0; temp[m] != '\0'; m++) push(stack, &top, temp[m]);
        }
       }
     }
     printt(stack, &top, inp, i);
   } while (inp[i] != '\0');
   if (strcmp(temp, "acc") == 0)
     printf(" \n accept the input ");
   else
     printf(" \n do not accept the input ");
}
void push(char* s, int* sp, char item) {
   if (*sp == 100)
     printf(" stack is full ");
   else {
     *sp = *sp + 1;

     s[*sp] = item;
   }
}
char stacktop(char* s) {
   char i;
   i = s[top];
   return i;
}
void isproduct(char x, char p) {
   int k, l;
   k = ister(x);
   l = isstate(p);
   strcpy(temp, A[l - 1].row[k - 1]);
}
int ister(char x) {
   int i;
   for (i = 0; i < 6; i++)
     if (x == ter[i]) return i + 1;

   return 0;
}
int isnter(char x) {
   int i;
   for (i = 0; i < 3; i++)
     if (x == nter[i]) return i + 1;

   return 0;
```

```
}
int isstate(char p) {
  int i;
  for (i = 0; i < 12; i++)
    if (p == states[i]) return i + 1;
  return 0;
}
void error() {
  printf(" error in the input ");
  exit(0);
}
void isreduce(char x, char p) {
  int k, l;
  k = isstate(x);
  l = isnter(p);
  strcpy(temp, G[k - 1].r[l - 1]);
}

char pop(char* s, int* sp) {
  char item;
  if (*sp == -1)
    printf(" stack is empty ");
  else {
    item = s[*sp];
    *sp = *sp - 1;
  }
  return item;
}
void printt(char* t, int* p, char inp[], int i) {
  int r;
  printf("\n");
  for (r = 0; r <= *p; r++) rep(t, r);
  printf("\t\t\t");
  for (r = i; inp[r] != '\0'; r++) printf("%c", inp[r]);
}
void rep(char t[], int r) {
  char c;
  c = t[r];
  switch (c) {
    case 'a':
      printf("0");
      break;
    case 'b':
      printf("1");
      break;
    case 'c':
      printf("2");
      break;
```

```
    case 'd':
      printf("3");
      break;
    case 'e':
      printf("4");
      break;
    case 'f':
      printf("5");
      break;
    case 'g':
      printf("6");
      break;
    case 'h':
      printf("7");
      break;
    case 'm':
      printf("8");
      break;
    case 'j':
      printf("9");
      break;
    case 'k':
      printf("10");
      break;
    case 'l':
      printf("11");
      break;
    default:
      printf("%c", t[r]);
      break;
  }
}
```

Output


Select C:\Users\kumar\OneDrive\Desktop\Untitled1.exe
Enter the input :i*i
stack input
0 i*i$
0i5 *i$
0F3 *i$
0T2 *i$
0T2*7 i$
0T2*7i5 $
0T2*7F10 $
0T2 $
0E1 $
accept the input
Process exited after 86.68 seconds with return value 20
Press any key to continue . . .

LAB-10

Q.1) Write a C program to implement operator precedence parsing.

Source Code:

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

char stack[20],ip[20],opt[10][10][1],ter[10];
int i,j,k,n,top=0,col,row;
for(i=0;i<10;i++)
{
 stack[i]=NULL;
 ip[i]=NULL;
 for(j=0;j<10;j++)
 {
 opt[i][j][1]=NULL;
 }
}
printf("Enter the no.of terminals :\n");
scanf("%d",&n);
printf("\nEnter the terminals :\n");
scanf("%s",&ter);
printf("\nEnter the table values :\n");
for(i=0;i<n;i++)
{
 for(j=0;j<n;j++)
 {
 printf("Enter the value for %c %c:",ter[i],ter[j]);
 scanf("%s",opt[i][j]);
 }
}
printf("\n**** OPERATOR PRECEDENCE TABLE ****\n");
for(i=0;i<n;i++)
{
printf("\t%c",ter[i]);
}
printf("\n");
for(i=0;i<n;i++){printf("\n%c",ter[i]);
for(j=0;j<n;j++){printf("\t%c",opt[i][j][0]);}}
stack[top]='$';
printf("\nEnter the input string:");
scanf("%s",ip);
i=0;
printf("\nSTACK\t\t\tINPUT STRING\t\t\tACTION\n");
printf("\n%s\t\t\t%s\t\t\t",stack,ip);
```

```
while(i<=strlen(ip))
{
for(k=0;k<n;k++)
{
if(stack[top]==ter[k])
col=k;
if(ip[i]==ter[k])
row=k;
}
if((stack[top]=='$')&&(ip[i]=='$')){
printf("String is accepted\n");
break;}
else if((opt[col][row][0]=='<') ||(opt[col][row][0]=='='))
{ stack[++top]=opt[col][row][0];
stack[++top]=ip[i];
 printf("Shift %c",ip[i]);
 i++;
  }
else{
 if(opt[col][row][0]=='>')
{
while(stack[top]!='<'){--top;}
top=top-1;
printf("Reduce");
}
else
{
printf("\nString is not accepted");
break;
}
}
printf("\n");
for(k=0;k<=top;k++)
{
printf("%c",stack[k]);
}
printf("\t\t\t");
for(k=i;k<strlen(ip);k++){
printf("%c",ip[k]);
}
printf("\t\t\t");
```

Output

```
Enter the value for + $:>
Enter the value for * i:<
Enter the value for * +:>
Enter the value for * *:>
Enter the value for * $:>
Enter the value for $ i:<
Enter the value for $ +:<
Enter the value for $ *:<
Enter the value for $ $:a

**** OPERATOR PRECEDENCE TABLE ****
        i       +       *       $

i       e       >       >       >
+       <       >       <       >
*       <       >       >       >
$       <       <       <       a
Enter the input string:i*i

STACK                   INPUT STRING                    ACTION

$                       i*i                     Shift i
$<i                     *i                      Reduce
$                       *i                      Shift *
$<*                     i                       Shift i
$<*<i
String is not accepted
Process returned 23 (0x17)   execution time : 71.159 s
Press any key to continue.
```

Lab 11

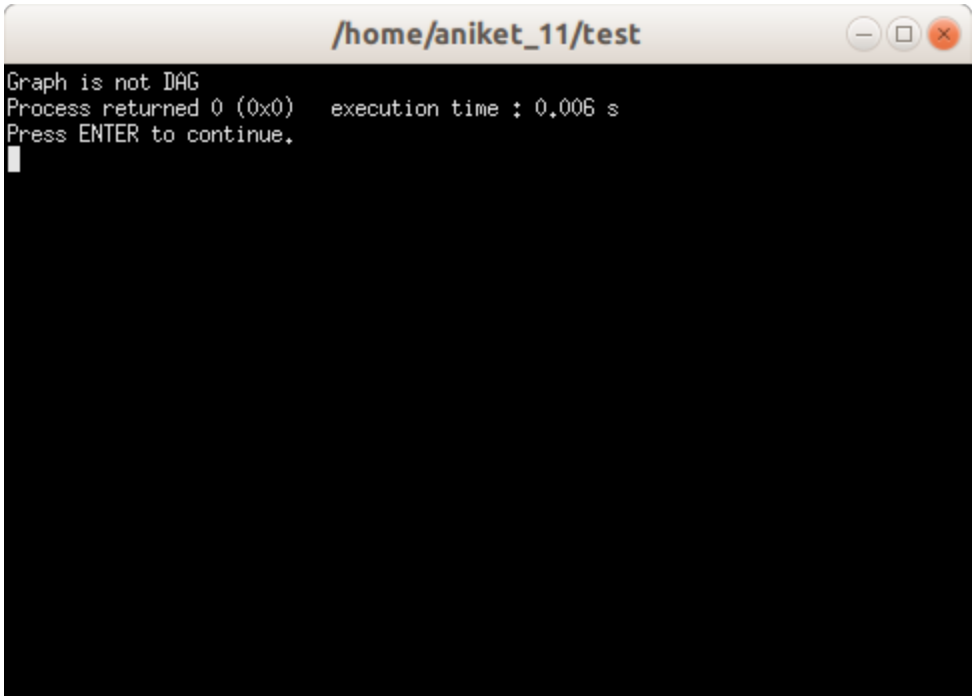
. Write a program to construct of DAG (Directed Acyclic Graph)

```
#include <iostream>
#include <vector>
using namespace std;

// data structure to store graph edges
struct Edge {
	int src, dest;
};

// class to represent a graph object
class Graph
{
public:
	// construct a vector of vectors to represent an adjacency list
	vector<vector<int>> adjList;

	// Graph Constructor
	Graph(vector<Edge> const &edges, int N)
	{
		// resize the vector to N elements of type vector<int>
		adjList.resize(N);

		// add edges to the Directed graph
		for (auto &edge: edges)
		{
			adjList[edge.src].push_back(edge.dest);
		}
	}
};

// Perform DFS on graph and set departure time of all
// vertices of the graph
int DFS(Graph const &graph, int v, vector<bool>
	&discovered, vector<int> &departure, int& time)
{
	// mark current node as discovered
	discovered[v] = true;

	// do for every edge (v -> u)
	for (int u : graph.adjList[v])
	{
		// u is not discovered
		if (!discovered[u])
			DFS(graph, u, discovered, departure, time);
	}
```

```
	// ready to backtrack
	// set departure time of vertex v
	departure[v] = time++;
}

// returns true if given directed graph is DAG
bool isDAG(Graph const& graph, int N)
{
	// stores vertex is discovered or not
	vector<bool> discovered(N);

	// stores departure time of a vertex in DFS
	vector<int> departure(N);

	int time = 0;

	// Do DFS traversal from all undiscovered vertices
	// to visit all connected components of graph
	for (int i = 0; i < N; i++)
		if (discovered[i] == false)
			DFS(graph, i, discovered, departure, time);

	// check if given directed graph is DAG or not
	for (int u = 0; u < N; u++)
	{
		// check if (u, v) forms a back-edge.
		for (int v : graph.adjList[u])
		{
			// If departure time of vertex v is greater
			// than equal to departure time of u, then
			// they form a back edge

			// Note that departure[u] will be equal to
			// departure[v] only if u = v i.e vertex
			// contain an edge to itself
			if (departure[u] <= departure[v])
				return false;
		}
	}

	// no back edges
	return true;
}

// Check if given digraph is a DAG (Directed Acyclic Graph) or not
int main()
{
```

```
// vector of graph edges as per above diagram
vector<Edge> edges =
{
        {0, 1}, {0, 3}, {1, 2}, {1, 3}, {3, 2}, {3, 4},
        {3, 0}, {5, 6}, {6, 3}
};

// Number of nodes in the graph
int N = 7;

// create a graph from given edges
Graph graph(edges, N);

// check if given directed graph is DAG or not
if (isDAG(graph, N))
        cout << "Graph is DAG";
else
        cout << "Graph is not DAG";

return 0;}
```

Output



LAB-12

Q.1) Write a C program to generate three address code.

Source Code:

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

void pm();
void plus();
void div();
int i, ch, j, l, addr = 100;
char ex[10], exp[10], exp1[10], exp2[10], id1[5], op[5], id2[5];
void main() {
  // clrscr();
  while (1) {
    printf(
        "\n1.assignment\n2.arithmetic\n3.relational\n4.Exit\nEnter the "
        "choice:");
    scanf("%d", &ch);
    switch (ch) {
      case 1:
        printf("\nEnter the expression with assignment operator:");
        scanf("%s", exp);
        l = strlen(exp);
        exp2[0] = '\0';
        i = 0;
        while (exp[i] != '=') {
          i++;
        }
        strncat(exp2, exp, i);
        strrev(exp);
        exp1[0] = '\0';
        strncat(exp1, exp, l - (i + 1));
        strrev(exp1);
        printf("Three address code:\ntemp=%s\n%s=temp\n", exp1, exp2);
        break;

      case 2:
        printf("\nEnter the expression with arithmetic operator:");
        scanf("%s", ex);
        strcpy(exp, ex);
        l = strlen(exp);
        exp1[0] = '\0';

        for (i = 0; i < l; i++) {
          if (exp[i] == '+' || exp[i] == '-') {
            if (exp[i + 2] == '/' || exp[i + 2] == '*') {
```

```
            pm();
            break;
          } else {
            plus();
            break;
          }
        } else if (exp[i] == '/' || exp[i] == '*') {
          div();
          break;
        }
      }
      break;

    case 3:
      printf("Enter the expression with relational operator");
      scanf("%s%s%s", &id1, &op, &id2);
      if (((strcmp(op, "<") == 0) || (strcmp(op, ">") == 0) ||
          (strcmp(op, "<=") == 0) || (strcmp(op, ">=") == 0) ||
          (strcmp(op, "==") == 0) || (strcmp(op, "!=") == 0)) == 0)
        printf("Expression is error");
      else {
        printf("\n%d\tif %s%s%s goto %d", addr, id1, op, id2, addr + 3);
        addr++;
        printf("\n%d\t T:=0", addr);
        addr++;
        printf("\n%d\t goto %d", addr, addr + 2);
        addr++;
        printf("\n%d\t T:=1", addr);
      }
      break;
    case 4:
      exit(0);
    }
  }
}
void pm() {
  strrev(exp);
  j = l - i - 1;
  strncat(exp1, exp, j);
  strrev(exp1);
  printf("Three address code:\ntemp=%s\ntemp1=%c%ctemp\n", exp1, exp[j + 1],
      exp[j]);
}
void div() {
  strncat(exp1, exp, i + 2);
  printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n", exp1, exp[i + 2],
      exp[i + 3]);
}
```

```
void plus() {
  strncat(exp1, exp, i + 2);
  printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n", exp1, exp[i + 2],
        exp[i + 3]);
}
```

```
1.assignment
2.arithmetic
3.relational
4.Exit
Enter the choice:2

Enter the expression with arithmetic operator:5-4*10
Three address code:
temp=4*10
temp1=5-temp
```