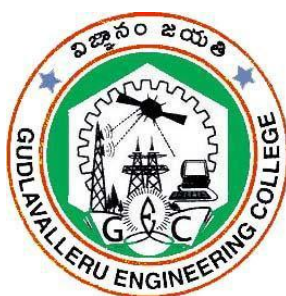


COMPUTER NETWORKS AND COMPILER DESIGN LAB

III Year – I Semester



Prepared by

Mr. J. N. V. R. Swarup Kumar
Assistant Professor

Mrs. D. Priyanka
Assistant Professor

Department of Computer Science and Engineering

GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institute with Permanent Affiliation to JNTUK, Kakinada)
Seshadri Rao Knowledge Village, Gudlavalleru – 521 356.

VISION & MISSION OF THE COLLEGE

Vision

To be a leading institution of engineering education and research, preparing students for leadership in their fields in a caring and challenging learning environment.

Mission

- To produce quality engineers by providing state-of-the-art engineering education.
- To attract and retain knowledgeable, creative, motivated and highly skilled individuals whose leadership and contributions uphold the college tenets of education, creativity, research and responsible public service.
- To develop faculty and resources to impart and disseminate knowledge and information to students and also to society that will enhance educational level, which in turn, will contribute to social and economic betterment of society.
- To provide an environment that values and encourages knowledge acquisition and academic freedom, making this a preferred institution for knowledge seekers.
- To provide quality assurance.
- To partner and collaborate with industry, government, and R and D institutes to develop new knowledge and sustainable technologies and serve as an engine for facilitating the nation's economic development.
- To impart personality development skills to students that will help them to succeed and lead.
- To instil in students the attitude, values and vision that will prepare them to lead lives of personal integrity and civic responsibility.
- To promote a campus environment that welcomes and makes students of all races, cultures and civilizations feel at home.
- Putting students face to face with industrial, governmental and societal challenges.

VISION & MISSION OF THE DEPARTMENT

Vision

To be a Centre of Excellence in Computer Science and Engineering education and training to meet the challenging needs of the industry and society.

Mission

- To impart quality education through well-designed curriculum in tune with the growing software needs of the industry.
- To serve our students by inculcating in them problem solving, leadership, teamwork skills and the value of commitment to quality, ethical behavior & respect for others.
- To foster industry-academia relationship for mutual benefit and growth

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

- Identify, analyze, formulate and solve Computer Science and Engineering problems both independently and in a team environment by using the appropriate modern tools.
- Manage software projects with significant technical, legal, ethical, social, environmental and economic considerations.
- Demonstrate commitment and progress in lifelong learning, professional development, leadership and communicate effectively with professional clients and the public.

PROGRAM OUTCOMES(POs):

Engineering Graduates will be able to:

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

6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

Students will be able to

- Design, develop, test and maintain reliable software systems and intelligent systems.
- Design and develop web sites, web apps and mobile apps.

Course Objectives

- To demonstrate the functionalities of various layers of OSI model.
- To demonstrate lexical analysis and syntax analysis phases of a compiler.

Learning Outcomes .

Upon successful completion of the course, the students will be able to

- implement data link layer framing and error detection methods.
- analyze the topological and routing strategies for an IP based networking infrastructure.
- develop code to implement lexical analyzer.
- implement lexical analyzer using LEX tool.
- implement parser using YACC tool.

Mapping of Course Outcomes with Program Outcomes:

CS2509 : COMPUTER NETWORKS AND COMPILER DESIGN LAB														
Course outcomes	Program Outcomes and Program Specific Outcome													
	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PS O 1	PS O 2
CO1:implement data link layer framing and error detection methods.	3	3	3		1			2	2	2		2		2
CO2:analyze the topological and routing strategies for an IP based networking infrastructure.	3	3	3		1			2	2	2		2		1 2
CO3:develop code to implement lexical analyzer.	3	3	3		2			2	2	2		2		2 2
CO4:implement lexical analyzer using LEX too	3	3	3		2			2	2	2		2		2 2
CO5:implement parser using YACC tool.	3	3	3		2			2	2	2		2		2 2
Computer Networks And Compiler Design Lab	3	3	3		3			3	3	3		3		3 3

INDEX

S.No	<u>TITLE</u>	Page No
1	System Requirements	8
2	Guidelines to Students	8
	<u>LIST OF EXPERIMENTS</u>	
	Computer Networks	
Exercise:1	Implement the data link layer framing methods such as character, character stuffing and bit stuffing	9
Exercise:2	Implement on a data set of characters using two CRC polynomials – CRC 12, CRC 16.	12
Exercise:3	Implement Dijkstra's algorithm to compute the shortest path through graph.	16
Exercise:4	Take an example subnet graph with weights indicating delay between nodes. Now obtain Routing table at each node using distance vector routing algorithm.	20
Exercise:5	Implement hierarchical routing algorithm.	22
Exercise:6	Implement error detecting techniques.	24
	Compiler Design	
Exercise:1	Implement DFA for the regular languages.	26
Exercise:2	Implement a PDA for context free languages.	28
Exercise:3	Implement a TM for phrase-structured languages.	31
Exercise:4	Design lexical analyzer to recognize the tokens and removes the comment lines and the blank spaces.	34
Exercise:5	Implement the lexical analyzer using LEX tool.	40
Exercise:6	Implement predictive parser for a given language	43
Exercise:7	Implement LALR bottom up parser for the given language.	49
Exercise:8	Implement the syntax analyzer using YACC tool.	53

S.No	<u>ADDITIONAL EXPERIMENTS</u>	Page No
	Computer Networks	
Exercise:1	Write a program to take an example subnet of hosts and obtain the broadcast tree for it.	56
Exercise:2	Write a program to implement caesar cipher substitution technique.	59
Exercise:3	Write a program to implement rail fence cipher transposition technique.	61
Exercise:4	Write a program to implement RSA algorithm to encrypt a text data and decrypt the same.	63

1. System Requirements

Recommended Systems / Software Requirements:

- Intel based desktop PC, ANSI C Compiler with Supporting Editors, IDE's such as Turbo C.
- Linux with gcc compiler
- Codetantra Platform for Computer Networks lab.

2. Guidelines to Students

- Equipment in the lab for the use of student community. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care. Any damage caused is punishable.
- Students are supposed to occupy the machines allotted to them and are not supposed to talk or make noise in the lab. The allocation is put up on the lab notice board.
- Lab can be used in free time / lunch hours by the students who need to use the systems should take prior permission from the lab in-charge.
- Lab records need to be submitted on or before date of submission.

Exercise:1a

Aim : Write a program to implement the data link layer framing method such as bit stuffing.

Description:

The new technique allows data frames to contain an arbitrary number of bits and allows character codes with an arbitrary no of bits per character. Each frame begins and ends with the special bit pattern, 01111110, called a flag byte. Whenever the sender's data link layer encounters five consecutive ones in the data, it automatically stuffs a 0 bit into the outgoing bit stream. This bit stuffing is analogous to character stuffing, in which a DLE is stuffed into the outgoing character stream before DLE in the data.

Program:

```
#include<stdio.h>
void main()
{
    int a[20],b[20],i,j,k,count,n;
    printf("Enter the frame size : ");
    scanf("%d",&n);
    printf("Enter the frame in the form of 0's and 1's : ");
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    i=0;
    count=1;
    j=0;
    while(i<n)
    {
        if(a[i]==1)
        {
            b[j]=a[i];
            for(k=i+1;a[k]==1&&k<n&&count<5;k++)
            {
                j++;
                b[j]=a[k];
                count++;
                if(count==5)
                {
                    j++;
                    b[j]=0;
                }
                i=k;
            }
        }
        else
        {
            b[j]=a[i];
        }
        i++;
        j++;
    }
}
```

```

    }
    printf("After bit stuffing : ");
    for(i=0;i<j;i++)
        printf("%d",b[i]);
    printf("\n");
}

```

Output:

Test Case - 1
User Output
Enter the frame size : 12
Enter the frame in the form of 0's and 1's : 0 1 0 1 1 1 1 1 1 0 0 1
After bit stuffing : 0101111101001

Test Case - 2
User Output
Enter the frame size : 5
Enter the frame in the form of 0's and 1's : 1 1 1 1 1
After bit stuffing : 111110

Test Case - 3
User Output
Enter the frame size : 6
Enter the frame in the form of 0's and 1's : 1 0 1 1 1 1
After bit stuffing : 101111

Exercise:1b

Aim: Write a program to implement the data link layer framing method such as character stuffing and also de stuff it.

Description:

In character stuffing, each frame starts with the ASCII character sequence DLE STX and ends with the sequence DLE ETX.(where DLE is Data Link Escape, STX is Start of TeXt and ETX is End of TeXt.) This method overcomes the drawbacks of the character count method. If the destination ever loses synchronization, it only has to look for DLE STX and DLE ETX characters. If however, binary data is being transmitted then there exists a possibility of the characters DLE STX and DLE ETX occurring in the data. Since this can interfere with the framing, a technique called character stuffing is used. The sender's data link layer inserts an ASCII DLE character just before the DLE character in the data. The receiver's data link layer removes this DLE before this data is given to the network layer. However character stuffing is closely associated with 8-bit characters and this is a major hurdle in transmitting arbitrary sized characters.

Program:

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int i,j,n,k,count,a[20],b[20];
    char d[20];

```

```

printf("Enter the number of characters : ");
scanf("%d",&n);
printf("Enter the characters : ");
scanf("%s",d);
printf("The original data : %s\n",d);
printf("The transmitted data : dlestd",d);
for(i=0;i<n;i++)
{
    j=i;
    if(d[j]=='d')
    {
        if(d[++j]=='l')
        {
            if(d[++j]=='e')
            {
                printf("dle");
            }
        }
    }
    printf("%c",d[i]);
}
printf("dlestd");
printf("\nThe received data : %s",d);
printf("\n");
}

```

Output:

Test Case - 1
User Output
Enter the number of characters : 9
Enter the characters : dledleabc
The original data : dledleabc
The transmitted data : dlestdledledledleabcdlestd
The received data : dledleabc

Test Case - 2
User Output
Enter the number of characters : 8
Enter the characters : abdlede
The original data : abdlede
The transmitted data : dlestdbledledledledlestd
The received data : abdlede

Viva Questions:

1. What is Framing?
2. What is Fixed Size Framing?
3. Define Character Stuffing?
4. What is Bit Stuffing?
5. Name the two sub layers of Data link layer. Specify their protocols.

Exercise:2a

Aim: Write a program to implement on a data set of characters the CRC encoding algorithm.

Description:

CRC (Cyclic Redundancy Check) is used for encoding the given bits using the generated string. The encoded information is transmitted to the other end. For encoding we perform the exclusive operation

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
    char msg[20],key[10],keycpy[20],temp[20],quot[20],rem[20];
    int msglen,keylen,i,j;
    printf("Enter the frame string : ");
    scanf("%s",msg);
    printf("Enter the generator string : ");
    scanf("%s",key);
    strcpy(keycpy,key);
    msglen=strlen(msg);
    keylen=strlen(key);
    for(i=0;i<keylen-1;i++)
    {
        msg[msglen+i]='0';
    }
    for(i=0;i<keylen;i++)
        temp[i]=msg[i];
    for(i=0;i<msglen;i++)
    {
        quot[i]=temp[0];
        if(quot[i]=='0')
            for(j=0;j<keylen;j++)
                key[j]='0';
        else
            for(j=0;j<keylen;j++)
                key[j]=keycpy[j];
        for(j=keylen-1;j>0;j--)
        {
            if(temp[j]==key[j])
                rem[j-1]='0';
            else
                rem[j-1]='1';
        }
        rem[keylen-1]=msg[i+keylen];
        strcpy(temp,rem);
    }
    strcpy(rem,temp);
    printf("The data to be transmitted is : ");
    for(i=0;i<msglen;i++)
```

```

        printf("%c",msg[i]);
    for(i=0;i<keylen-1;i++)
        printf("%c",rem[i]);
    printf("\n");
}

```

Output:

Test Case - 1
User Output
Enter the frame string : 11101100
Enter the generator string : 1101
The data to be transmitted is : 11101100110

Test Case - 2
User Output
Enter the frame string : 100100
Enter the generator string : 1101
The data to be transmitted is : 100100001

Test Case - 3
User Output
Enter the frame string : 11010011101100
Enter the generator string : 1011
The data to be transmitted is : 11010011101100100

Exercise:2b

Aim: Write a program to implement on a data set of characters the CRC decoding algorithm.

Description:

CRC (Cyclic Redundancy Check) is used for decoding the string which is encoded in the CRC encoding algorithm. Also we check for the errors that occur in the transmitted data. For this we use exclusive operation.

Program:

```

#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
    int p[20],g[20],i,j,k;
    char cp[20],cg[20];
    printf("Enter the received string : ");
    scanf("%s",cp);
    printf("Enter the polynomial string : ");
    scanf("%s",cg);
    for(i=0;i<strlen(cp);i++)
        p[i]=cp[i]-'0';

```

```

for(i=0;i<strlen(cg);i++)
    g[i]=cg[i]-'0';
i=0;
j=0;
while(1)
{
    j=0;
    while(p[j]!=1)
        j++;
    if(j>=strlen(cp)-(strlen(cg)-1))
        break;
    k=0;
    for(i=j;i<j+strlen(cg);i++)
        p[i]=p[i]^g[k];
}
for(i=0;i<strlen(cp);i++)
{
    if(p[i]!=0)
        break;
}
if(i!=strlen(cp))
{
    printf("No error\n");
    printf("Data received is : ");
    for(i=0;i<strlen(cp)-(strlen(cg)-1);i++)
    {
        cp[i]=cp[i]-'0';
        printf("%d",cp[i]);
        cp[i]=cp[i]-'1';
    }
}
else
    printf("Error");
printf("\n");
}

```

Output:

Test Case - 1
User Output
Enter the received string : 11010110111110
Enter the polynomial string : 10011
No error
Data received is : 1101011011

Test Case - 2
User Output
Enter the received string : 100100001
Enter the polynomial string : 1101
No error
Data received is : 100100

Test Case - 3
User Output
Enter the received string : 100000001
Enter the polynomial string : 1101
Error

Viva Questions:

1. What are the types of errors?
2. What is Error Detection? What are its methods?
3. What is Redundancy?
4. What is VRC?
5. What is LRC?
6. What is CRC?
7. What is Checksum?
8. List the steps involved in creating the checksum.

Exercise:3

Aim: Write a program on Implementation of Dijkstra's shortest path algorithm.

Description:

Let the node at which we are starting be called the initial node. Let the distance of node Y be the distance from the initial node to Y. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

1. Mark all nodes unvisited. Create a set of all the unvisited nodes called the unvisited set.
2. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes. Set the initial node as current.
3. For the current node, consider all of its unvisited neighbours and calculate their tentative distances through the current node. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbour B has length 2, then the distance to B through A will be $6 + 2 = 8$. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.
4. When we are done considering all of the unvisited neighbours of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3. When planning a route, it is actually not necessary to wait until the destination node is "visited" as above: the algorithm can stop once the destination node has the smallest tentative distance among all "unvisited" nodes (and thus could be selected as the next "current").

Program:

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10
void dij(int g[MAX][MAX],int n,int stnode);
int main()
{
    int g[MAX][MAX],i,j,n,u;
    printf("Enter number of vertices : ");
    scanf("%d",&n);
    printf("Enter the adjacency matrix\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%d",&g[i][j]);
    printf("Enter the starting node : ");
    scanf("%d",&u);
    dij(g,n,u);
    return 0;
}
```



```

void dij(int g[MAX][MAX],int n,int stnode)
{
    int cost[MAX][MAX],dist[MAX],pred[MAX];
    int vis[MAX],count,mindist,nextnode,i,j;
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            if(g[i][j]==0)
                cost[i][j]=INFINITY;
            else
                cost[i][j]=g[i][j];
        }
    }
    for(i=0;i<n;i++)
    {
        dist[i]=cost[stnode][i];
        pred[i]=stnode;
        vis[i]=0;
    }
    dist[stnode]=0;
    vis[stnode]=1;
    count=1;
    while(count<n-1)
    {
        mindist=INFINITY;
        for(i=0;i<n;i++)
        {
            if(dist[i]<mindist&&!vis[i])
            {
                mindist=dist[i];
                nextnode=i;
            }
        }
        vis[nextnode]=1;
        for(i=0;i<n;i++)
        {
            if(!vis[i])
            {
                if(mindist+cost[nextnode][i]<dist[i])
                {
                    dist[i]=mindist+cost[nextnode][i];
                    pred[i]=nextnode;
                }
            }
        }
        count++;
    }
    for(i=0;i<n;i++)
    {

```

```

    if(i!=stnode)
    {
        printf("Distance of node %d = %d\n",i,dist[i]);
        printf("Path = %d",i);
        j=i;
        do
        {
            j=pred[j];
            printf(" <- %d",j);
        }while(j!=stnode);
        printf("\n");
    }
}
}

```

Output:

Test Case - 1	
User Output	
Enter number of vertices : 5	
Enter the adjacency matrix 0 10 0 30 100	
10 0 50 0 0	10 0 50 0 0
0 50 0 20 10	0 50 0 20 10
30 0 20 0 60	30 0 20 0 60
100 0 10 60 0	100 0 10 60 0
Enter the starting node : 0	
Distance of node 1 = 10	
Path = 1 <- 0	
Distance of node 2 = 50	
Path = 2 <- 3 <- 0	
Distance of node 3 = 30	
Path = 3 <- 0	
Distance of node 4 = 60	
Path = 4 <- 2 <- 3 <- 0	

Test Case - 2	
User Output	
Enter number of vertices : 3	
Enter the adjacency matrix 10 20 0	
20 10 40	20 10 40
60 40 30	60 40 30
Enter the starting node : 0	
Distance of node 1 = 20	
Path = 1 <- 0	
Distance of node 2 = 60	
Path = 2 <- 1 <- 0	

Viva Questions:

1. What is Protocol?
2. What are the key elements of protocols?
3. What are the key design issues of a computer Network?
4. Define Bandwidth and Latency?
5. Define Routing?
6. What are the responsibilities of Network Layer?
7. Dijkstra's Algorithm is used to solve _____ problems.
8. Which of the following is the most commonly used data structure for implementing Dijkstra's Algorithm?
9. What is the time complexity of Dijkstra's algorithm?
10. Dijkstra's Algorithm cannot be applied on _____

Exercise:4

Aim: Take an example subnet graph with weights indicating delay between nodes. Now obtain routing table at each node using distance vector routing algorithm.

Description:

A router transmits its distance vector to each of its neighbours in a routing packet. Each router receives and saves the most recently received distance vector from each of its Neighbours. A router recalculates its distance vector when:

1. It receives a distance vector from a neighbour containing different information than before.
2. It discovers that a link to a neighbour has gone down.

Program:

```
#include<stdio.h>
struct node
{
    unsigned dist[20];
    unsigned from[20];
}rt[10];
int main()
{
    int cost[10][10],n,i,j,k,count=0;
    printf("Enter the number of nodes : ");
    scanf("%d",&n);
    printf("Enter the cost matrix\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&cost[i][j]);
            cost[i][i]=0;
            rt[i].dist[j]=cost[i][j];
            rt[i].from[j]=j;
        }
    }
    do
    {
        count=0;
        for(i=0;i<n;i++)
            for(j=0;j<n;j++)
                for(k=0;k<n;k++)
                    if(rt[i].dist[j]>cost[i][k]+rt[k].dist[j])
                    {
                        rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
                        rt[i].from[j]=k;
                        count++;
                    }
    }while(count!=0);
    for(i=0;i<n;i++)
    {
```

```

    printf("For router %d\n",i+1);
    for(j=0;j<n;j++)
    {
        printf("Node %d via %d distance
%d\n",j+1,rt[i].from[j]+1,rt[i].dist[j]);
    }
    printf("\n");
}

```

Output:

Test Case - 1	
User Output	
Enter the number of nodes : 3	
Enter the cost matrix 0 2 4	
2 0 5	2 0 5
4 5 0	4 5 0
For router 1	
Node 1 via 1 distance 0	
Node 2 via 2 distance 2	
Node 3 via 3 distance 4	
For router 2	
Node 1 via 1 distance 2	
Node 2 via 2 distance 0	
Node 3 via 3 distance 5	
For router 3	
Node 1 via 1 distance 4	
Node 2 via 2 distance 5	
Node 3 via 3 distance 0	

Test Case - 2	
User Output	
Enter the number of nodes : 3	
Enter the cost matrix 0 1 5	
1 0 2	1 0 2
5 2 0	5 2 0
For router 1	
Node 1 via 1 distance 0	
Node 2 via 2 distance 1	
Node 3 via 2 distance 3	
For router 2	
Node 1 via 1 distance 1	
Node 2 via 2 distance 0	
Node 3 via 3 distance 2	
For router 3	
Node 1 via 2 distance 3	
Node 2 via 2 distance 2	
Node 3 via 3 distance 0	

Viva Questions:

1. What is Distance-Vector Routing Protocol?
2. What is Routing table?

Exercise:5

Aim: Implement hierarchical routing algorithm.

Description:

This is essentially a 'Divide and Conquer' strategy. The network is divided into different regions and a router for a particular region knows only about its own domain and other routers.

Thus, the network is viewed at two levels:

1. The Sub-network level, where each node in a region has information about its peers in the same region and about the region's interface with other regions. Different regions may have different 'local' routing algorithms. Each local algorithm handles the traffic between nodes of the same region and also directs the outgoing packets to the appropriate interface.
2. The Network Level, where each region is considered as a single node connected to its interface nodes. The routing algorithms at this level handle the routing of packets between two interface nodes, and is isolated from intra-regional transfer.

Networks can be organized in hierarchies of many levels; e.g. local networks of a city at one level, the cities of a country at a level above it, and finally the network of all nations.

In Hierarchical routing, the interfaces need to store information about:

- i. All nodes in its region which are at one level below it.
- ii. Its peer interfaces.
- iii. At least one interface at a level above it, for outgoing packages

Program:

```
#include<stdio.h>
#include<conio.h>
struct full
{
    char line[10],dest[10];
    int hops;
}f[20];
void main()
{
    int nv,i,min,minver;
    char sv[20],temp;
    printf("Enter number of vertices : ");
    scanf("%d",&nv);
    printf("Enter source vertex : ");
    scanf("%s",sv);
    printf("Enter full table for source vertex %s\n",sv);
    for(i=0;i<nv;i++)
        scanf("%s%s%d",f[i].dest,f[i].line,&f[i].hops);
    printf("HIERARCHICAL TABLE");
    for(i=0;i<nv;)
    {
        if(sv[0]==f[i].dest[0])
        {
            printf("\n%s %s %d",f[i].dest,f[i].line,f[i].hops);
            i++;
        }
    }
}
```

```

else
{
    min=1000;
    minver=0;
    temp=f[i].dest[0];
    while(temp==f[i].dest[0])
    {
        if(f[i].hops<min)
        {
            minver=i;
        }
        i++;
    }
    printf("\n%c %s %d",temp,f[minver].line,f[minver].hops);
}
}
printf("\n");
}

```

Output:

Test Case - 1	
User Output	
Enter number of vertices : 4	
Enter source vertex : 1a	
Enter full table for source vertex 1a 1a 0 0	
1b 1c 2	1b 1c 2
1c 1d 4	1c 1d 4
1a 1d 5	1a 1d 5
HIERARCHICAL TABLE	
1a 0 0	
1b 1c 2	
1c 1d 4	
1a 1d 5	

Viva Questions:

1. What is static and dynamic routing?
2. What are the fields included in routing table?
3. Difference between Hierarchical and Flat Routing Protocol.

Exercise:6

Aim: Write a program to implement one of the error detection technique , simple parity check for even parity.

Description:

Blocks of data from the source are subjected to a check bit or parity bit generator form, where a parity of :

1 is added to the block if it contains odd number of 1's, and

0 is added if it contains even number of 1's

This scheme makes the total number of 1's even, that is why it is called even parity checking.

Program:

```
#include<stdio.h>
#include<string.h>
void main()
{
    int count=0,i=-1,e;
    char a[10],b[10],str1[]="0",str2[]="1";
    printf("Enter the data word : ");
    gets(a);
    strcpy(b,a);
    if(strlen(a)>4)
        printf("Please enter only 4 bits\n");
    else
    {
        printf("1.Without error 2.Error\n");
        scanf("%d",&e);
        if(e==1)
        {
            for(i=0;i<=4;i++)
            {
                if(a[i]=='1')
                    count+=1;
            }
            if(count%2==0)
                strcat(a,str1);
            else
                strcat(a,str2);
            printf("After adding the parity bit : %s",a);
            printf("\nSyndrome checking S value : 0 and no error");
            printf("\nThe data word : %s\n",b);
        }
        else
        {
            strcat(a,str1);
            printf("After adding the parity bit : %s",a);
            printf("\nSyndrome checking S value : 1 and error occur\n");
        }
    }
}
```


Output:

Sample Input and Output - 1:

```
Enter the data word : 1011
1.Without error 2.Error
After adding the parity bit : 10111
Syndrome checking S value : 0 and no error
The data word : 1011
```

Sample Input and Output - 2:

```
Enter the data word : 1011
1.Without error 2.Error
After adding the parity bit : 10110
Syndrome checking S value : 1 and error occur
```

Sample Input and Output - 2:

```
Enter the data word : 101010
Please enter only 4 bits
```

Viva Questions:

1. Define parity check.

Exercise:1

Aim: Implement DFA for the regular languages.

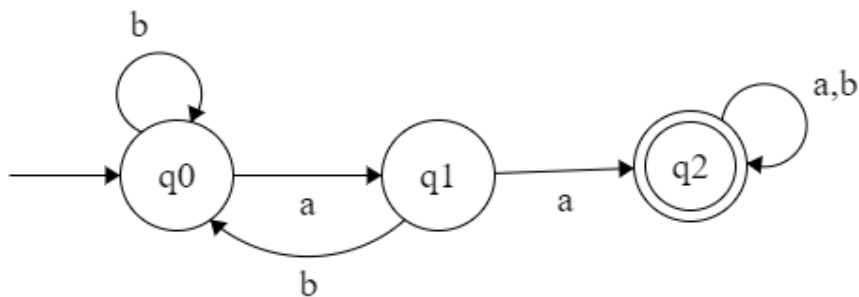
Description:

A deterministic finite automaton (DFA) is a finite-state machine that accepts or rejects strings of symbols and only produces a unique computation of the automaton for each input string.

A deterministic finite automaton M , is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite set of states.
- Σ is a finite input alphabet.
- δ is the transition function mapping $Q \times \Sigma \rightarrow Q$
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is the set of final states.

a) A set of all strings which contains 'aa' as substring over {a,b}.

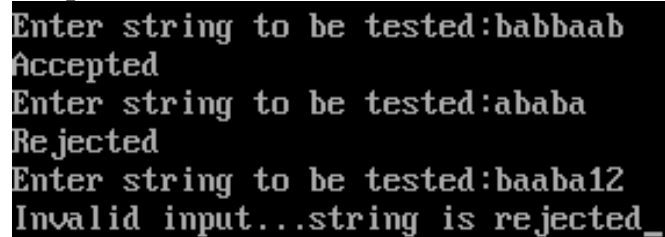
**Program:**

```

#include<stdio.h>
main()
{
  int i,st=0;
  char str[10],ch;
  clrscr();
  printf("\nEnter string to be tested:");
  gets(str);
  for(i=0;str[i]!='\0';i++)
  {
    ch=str[i];
    if(ch=='a'||ch=='b')
    {
      switch(st)
      {
        case 0:if(ch=='b')
                st=0;
              else if(ch=='a')
                st=1;
                break;
        case 1:if(ch=='b')
                st=0;
              else if(ch=='a')
                st=2;
                break;
      }
    }
  }
}

```

```
case 2:if(ch=='a' || ch=='b')
    st=2;
    break;
default:break;
}
}
else
{
    printf("Invalid input...string is rejected");
    exit(0);
}
}
if(st==2)
printf("Accepted");
else
printf("Rejected");
getch();
}
```

Output:

```
Enter string to be tested:babbaab
Accepted
Enter string to be tested:ababa
Rejected
Enter string to be tested:baaba12
Invalid input...string is rejected_
```

Viva Questions:

1. What are the types of finite automata?
2. What is DFA?
3. What is Regular Expression?
4. What is Regular Language?
5. What are the limitations of finite automata?

Exercise:2

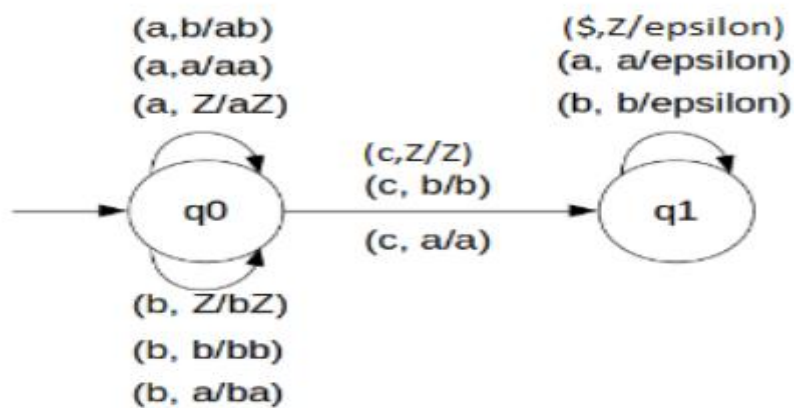
Aim: Implement a PDA for context free languages.

Description:

A *pushdown automaton* M is a system $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where

- 1) Q is a finite set of states;
- 2) Σ is an alphabet called the *input alphabet*;
- 3) Γ is an alphabet, called the *stack alphabet*;
- 4) q_0 in Q is the *initial state*;
- 5) Z_0 in Γ is a particular stack symbol called the *start symbol*;
- 6) $F \subseteq Q$ is the set of *final states*;
- 7) δ is a mapping from $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$.

a) Design PDA for the language $L = \{wcw^r \mid w \in \{a,b\}^*\}$.

**Program:**

```
#include<stdio.h>
char stack[20];
int top=-1;
void push(char var)
{
    stack[++top]=var;
}
char pop()
{
    char n;
    n=stack[top--];
    return n;
}
void main()
{
    char ch,str[10];
    int i,state=0;
    clrscr();
    printf("Enter string to be tested:");
    gets(str);
    push('z');
    for(i=0;str[i]!='\0';i++)
    {
        ch=str[i];
```

```

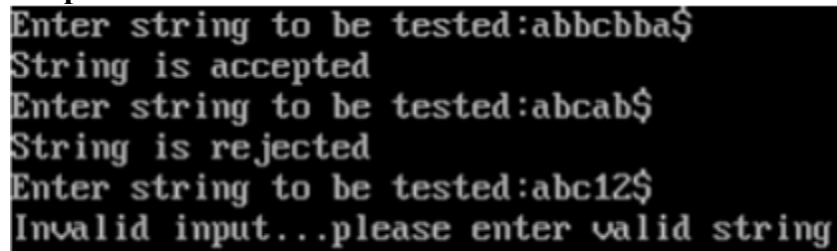
if(ch=='a'||ch=='b'||ch=='$'||ch=='c')
{
    switch(state)
    {
        case 0:if((ch=='a'||ch=='b')&&stack[top]=='z')
            {
                push(ch);
                state=0;
            }
            else
            if((ch=='a'||ch=='b')&&(stack[top]=='a'||stack[top]=='b'))
            {
                push(ch);
                state=0;
            }
            else
            if(ch=='c'&&(stack[top]=='a'||stack[top]=='b'||stack[top]=='z'))
            {
                state=1;
            }

            break;
        case 1: if(ch=='a'&& stack[top]=='a')
            {
                pop();
                state=1;
            }
            else if(ch=='b'&& stack[top]=='b')
            {
                pop();
                state=1;
            }
            else if(ch=='$' && stack[top]=='z')
            {
                pop();
                state=1;
            }
            break;
        default: break;
    }
}
else
{
    printf("Invalid input...please enter valid string");
    exit(0);
}

if(stack[top]=='\0')
printf("String is accepted");

```

```
else
printf("String is rejected");
getch();
}
```

Output:

```
Enter string to be tested:abbcbbba$
String is accepted
Enter string to be tested:abcbab$
String is rejected
Enter string to be tested:abc12$
Invalid input...please enter valid string
```

Viva Questions:

1. What is PDA?
2. What is context free language?
3. What is context free grammar?
4. What is derivation tree?
5. What is ambiguous grammar?

Exercise:3

Aim: Implement a TM for phrase-structured languages.

Description:

A Turing machine (TM) is denoted by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F),$$

where

Q is the finite set of *states*,

Γ is the finite set of allowable *tape symbols*,

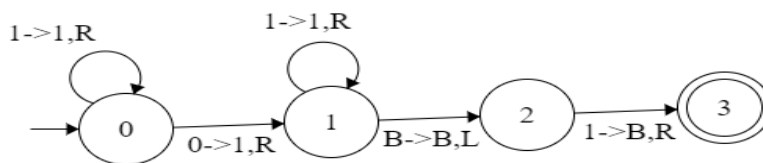
B , a symbol of Γ , is the *blank*,

Σ , a subset of Γ not including B , is the set of *input symbols*,

δ is the *next move function*, a mapping from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$ (δ may, however, be undefined for some arguments),

q_0 in Q is the *start state*,

$F \subseteq Q$ is the set of *final states*.

a) TM for addition of two unary numbers.**Program:**

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

```
void main()
```

```
{
```

```
    char str[20],c;
```

```
    int i=0,st=0;
```

```
    clrscr();
```

```
    printf("Enter string:");
```

```
    gets(str);
```

```
    while(str[i]!='\0')
```

```
    {
```

```
        c=str[i];
```

```
        if(c=='0'||c=='1'||c=='B')
```

```
        {
```

```
            switch(st)
```

```
            {
```

```
                case 0:if(c=='1')
```

```
                {
```

```
                    str[i]='1';
```

```
                    i++;
```

```
                    st=0;
```

```
                }
```

```
                else if(c=='0')
```

```
                {
```

```
                    str[i]='1';
```

```
                    i++;
```

```
                    st=1;
```

```
                }
```

```

        break;
        case 1:if(c=='1')
        {
            str[i]='1';
            i++;
            st=1;
        }
        else if(c=='B')
        {
            str[i]='B';
            i--;
            st=2;
        }
        break;
        case 2:if(c=='1')
        {
            str[i]='B';
            i++;
            st=3;
        }
        break;
        case 3:if(c=='B')
        {
            str[i]='B';
            i++;
            st=3;
        }
        break;
        default: break;
    }
}
else
{
    printf("Invalid input... please enter valid string");
    exit(0);
}
}
if(st==3)
{
    printf("Halt and accept");
    printf("\nAddition of two unary numbers is:");
    puts(str);
}
else
    printf("Halt and reject");
getch();
}

```


Output:

```
Enter string:11011B
Halt and accept
Addition of two unary numbers is:1111BB

Enter string:101
Halt and reject
Enter string:11012B
Invalid input... please enter valid string_
```

Viva Questions:

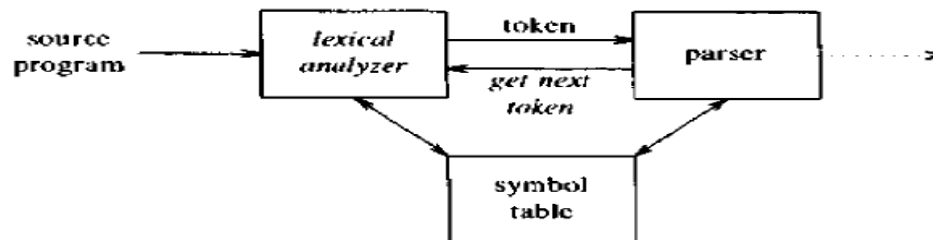
1. What is Turing Machine?
2. What is recursive language?
3. What is recursively enumerable language?
4. What are decidable problems?

Exercise:4

Aim: Design lexical analyzer to recognize the tokens and removes the comment lines and the blank spaces.

Description:

Lexical analysis is the first phase of a compiler. The lexical analysis phase reads the characters from the source program and group them into stream of tokens. Common token names are identifiers, keywords, separator (also known as punctuators), operators, literals, comments.

**Algorithm:**

1. Declare an array and store the keywords in that array.
2. Take a file as an input.
3. Read the characters from the file (sequence of characters is string) till the end of the file and recognize the tokens.
 - a. If the string matches with any of the keywords, print that string is a keyword.
 - b. If the string is not a keyword, print it as an identifier.
 - c. If the string is a number, print it as a digit.
 - d. If the string matches with operator and special symbols, print the corresponding message.

Program:

```

#include<stdio.h>
#include<ctype.h>
#include<string.h>
int lineno=0;
struct
{
  char token[10];
}symbol_table[50];
int loc=0;
int kcnt=32;

char keylist[][20]={"auto","break","case","continue","const","char","double",
  ,"default","do","extern","enum","else","float","for","goto","int","if",
  ,"long","return","register","static","short","switch","signed","struct",
  ,"sizeof","typedef","unsigned","union","void","volatile","while"};

char lexbuf[100],c;
int tokenval,i,p;
void comment(char);
void digit(char);
void id(char);
  
```

```
void print_symbol_table();
int insert(char *);
int lookup(char *);
void string(char);
FILE *fp;
void main()
{
    int st;
    char c,filename[10];
    clrscr();
    printf("Enter filename:");
    gets(filename);
    fp=fopen(filename,"r");

    while((c=fgetc(fp))!=EOF)
    {
        st=0;
        if(c==' ' || c=='\t');
        else if(c=='\n')
        {
            printf("\n");
            lineno++;
        }
        else if(c=='/')
        {
            st=1;
        }
        else if (isdigit(c))
        {
            st=2;
        }
        else if (isalpha(c))
        {
            st=3;
        }
        else if(c=="")
        {
            st=4;
        }
        else
        {
            switch(c)
            {
                case '+':
                case '-':
                case '*':
                case '/':
                case '%':
                case '=':
```

```

                case '<':
                case '>':printf("<%c,%s>",c,"operator");
                        break;
        }
}

switch(st)
{
case 1:comment(c);
        break;
case 2:digit(c);
        break;
case 3:id(c);
        break;
case 4:string(c);
        break;
}
}
print_symbol_table();
printf("\n No of lines = %d",lineno);
getch();
}

void comment(char c)
{
        c=fgetc(fp);
        if(c=='/')
        {
                printf("Single line comment");
                do
                {
                        c=fgetc(fp);
                }while((c=fgetc(fp))!='\n');
                return;
        }
        else if(c=='*')
        {
                while(1)
                {
                        c=fgetc(fp);
                        if(c=='*')
                        {
                                c=fgetc(fp);
                                if(c=='/')
                                printf("comment");
                                break;
                        }
                }
        }
}

```

```

                else continue;
            }
        }
    }

void digit(char c)
{
    tokenval = c-'0';
    c=fgetc(fp);
    while(isdigit(c)) {
        tokenval = tokenval*10 + c-'0';
        c=fgetc(fp);
    }

    ungetc(c,fp);
    printf("digit=%d",tokenval);
    return;
}

void id(char c)
{
    i=0;
    while(isalnum(c))
    {
        lexbuf[i++]=c;
        c=fgetc(fp);
    }
    lexbuf[i]='\0';
    ungetc(c,fp);

    for(i=0;i<kcnt;i++)
    {
        if(strcmp(lexbuf,keylist[i])==0)
        {
            printf("<%s,%s>",lexbuf,"keyword");
            break;
        }
    }
    if(i==kcnt)
    {
        printf("<%s,%s>",lexbuf,"id");
        p=lookup(lexbuf);
        if(p==-1)
            p=insert(lexbuf);
    }
}

void string(char c)
{
    do

```

```
        {
            c=fgetc(fp);
        }while(c!="");
        printf("String literal");
    }

int insert(char *lexeme)
{
    strcpy(symbol_table[loc].token,lexeme);
    return(loc++);
}

int lookup(char *s)
{
    int i;
    for(i=0;i<loc;i++)
        if (strcmp(s,symbol_table[i].token)==0)
            return (i);
    return -1;
}

void print_symbol_table()
{
    int i;
    printf("\n\nloc - symbol \n");
    for(i=0;i<loc;i++)
        printf("\n %d - %s",i,symbol_table[i].token);
}
```

Output:**“sample.c”**

```
main()
{
    int a=10,b=20;
    int c;
    c=a+b;
    printf("%d",c);
    getch();
}
```

```
Enter filename:sample.c
<main,id>

<int,keyword><a,id><=,operator>digit=10<b,id><=,operator>digit=20
<int,keyword><c,id>
<c,id><=,operator><a,id><+,operator><b,id>
<printf,id>String literal<c,id>
<getch,id>

loc - symbol
0 - main
1 - a
2 - b
3 - c
4 - printf
5 - getch
No of lines = 8_
```

Viva Questions:

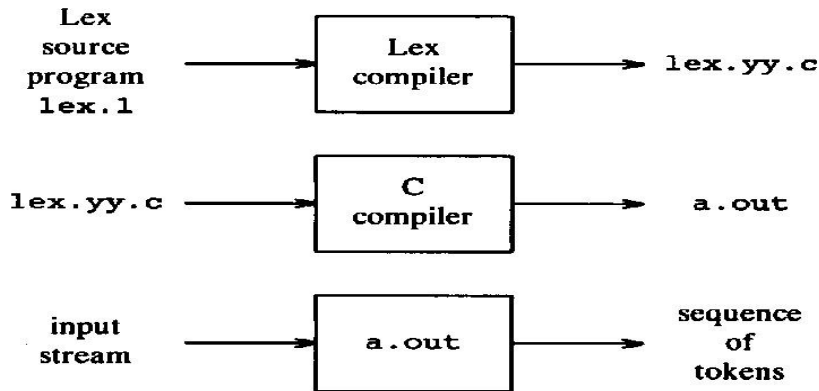
1. What is a token?
2. What are the phases of compiler?
3. What is lexical analysis?
4. What is a symbol table?
5. What is the difference between compiler and interpreter?

Exercise:5

Aim: Implement the lexical analyzer using LEX tool.

Description:

- LEX Tool is widely used to specify lexical analyzers for a variety of languages.
- The tool is referred as LEX Compiler and its input specification as the LEX Language.

**Lex Specifications:**

- A Lex program consists of three parts:
 - declarations
% %
 - translation rules
% %
 - auxiliary procedures
- The **Declarations section** includes declarations of variables, manifest constants, and regular definitions.
- The regular definitions are used as components of the regular expressions appearing in the translation rules.
- The **Translation Rules** of a Lex program are statements of the form


```

P1          {action1}
P2          {action2}
.....
Pn          {actionn}
      
```

 where each P₁ is a regular expression and each action₁ is a program fragment describing what action the lexical analyzer should take when pattern P_i matches a lexeme.
- In Lex, the actions are written in C; but they can be in any implementation language.
- The third section holds whatever **auxiliary procedures** are needed by the actions.
- These procedures can be compiled separately and loaded with the lexical analyzer.

Program:

```

% {
/* program to recognize a c program */
% }
identifier [a-zA-Z][a-zA-Z0-9]*
%%
  
```



```

#.* { printf("\n%s is a PREPROCESSOR DIRECTIVE",yytext);}
int |
float |
char |
double |
while |
for |
do |
if |
break |
continue |
void |
switch |
case |
long |
struct |
const |
typedef |
return |
else |
goto {printf("\n\t%s is a KEYWORD",yytext);}
{identifier}\( {printf("\n\nFUNCTION\n\t%s",yytext);}
\{ { printf("\n BLOCK BEGINS");}
\} { printf("\n BLOCK ENDS"); }
{identifier}\([0-9]*\)? { printf("\n %s IDENTIFIER",yytext);}
\."*\ " { printf("\n\t%s is a STRING",yytext);}
[0-9]+ { printf("\n\t%s is a NUMBER",yytext);}
= {printf("\n\t%s is an ASSIGNMENT OPERATOR",yytext);}
<= |
>= |
< |
== |
> { printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}
%%
int main(int argc,char **argv)
{
if (argc > 1)
{
FILE *file;
file = fopen(argv[1],"r");
if(!file)
{
printf("could not open %s \n",argv[1]);
exit(0);
}
yyin = file;
}
yylex();
printf("\n\n");
return 0;
}

```

```

}
int yywrap()
{
return 0;
}

```

Output:

```

"l1.1" 28L, 808C written
[cse17_e2@localhost ~]$ lex l1.1          15,23      All
[cse17_e2@localhost ~]$ cc lex.yy.c
[cse17_e2@localhost ~]$ ./a.out
#include<stdio.h>
#include<stdio.h> is a preprocessor directive

main()
main() is a function

{
Block begins
int a=10,b=20;
int is a keyword

a is an identifierAssignment operator10 is a digit
;
b is an identifierAssignment operator20 is a digit
;
printf("Sample lex program");

printf is an identifier(
"Sample lex program" is literal
);
}
Block ends

```

Viva Questions:

1. What is regular expression?
2. What is the regular expression for identifiers?
3. List the different sections available in LEX compiler?
4. What is an auxiliary definition?
5. How can we define the translation rules?

Exercise:6

Aim: Implement predictive parser for a given language.

$$E \rightarrow TE'$$

$$E' \rightarrow +TE' \mid \epsilon$$

$$T \rightarrow FT'$$

$$T' \rightarrow *FT' \mid \epsilon$$

$$F \rightarrow (E) \mid id$$
Description:

Predictive parsing is one of the top-down parsing technique. In this, the parse table is used with input. The parse table shouldn't contain multiple entries. The grammar which is acceptable by this parser is LL(1) grammar. There is no backtracking in predictive parsing.

Algorithm:

Input: A string w and a parsing table M for grammar G .

Output: If w is in $L(G)$, a leftmost derivation of w ; otherwise, an error indication.

Method: Initially, the parser has $\$S$ on the stack with S , the start symbol of G on top, and $w\$$ in the input buffer. The program that utilizes the predictive parsing table M to produce a parse for the input is as follows:

set ip to point to the first symbol of $w\$$;

repeat

begin

let X be the top stack symbol and a the symbol pointed to by ip ;

if X is a terminal or $\$$ then

if $X=a$

then pop X from the stack and advance ip

else

error()

else/* X is a nonterminal*/

if $M[X,a]=X \rightarrow Y_1, Y_2, \dots, Y_k$ then

begin

pop X from the stack;

push Y_k, Y_{k-1}, \dots, Y_1 on to the stack, Y_1 on top

end

else

error()

end/*first begin end*/

until $X=\$$

Predictive parsing table :

NON-TERMINAL	id	+	*	()	\$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$		

Program:

```
#include<stdio.h>
int stack[20],top=-1;
void push(int item)
{
    if(top>=20)
    {
        printf("STACK OVERFLOW");
        exit(1);
    }
    stack[++top]=item;
}
int pop()
{
    int ch;
    if(top<=-1)
    {
        printf("underflow");
        exit(1);
    }
    ch=stack[top--];
    return ch;
}
char convert(int item)
{
    char ch;
    switch(item)
    {
        case 0:return('E');
        case 1:return('e');
        case 2:return('T');
        case 3:return('t');
        case 4:return('F');
        case 5:return('i');
```

```
    case 6:return('+');
    case 7:return('*');
    case 8:return('(');
    case 9:return(')');
    case 10:return('$');
}
}
void main()
{
    int m[10][10],i,j,k;
    char ips[20];
    int ip[10],a,b,t;
    m[0][0]=m[0][3]=21;
    m[1][1]=621;
    m[1][4]=m[1][5]=-2;
    m[2][0]=m[2][3]=43;
    m[3][1]=m[3][4]=m[3][5]=-2;
    m[3][2]=743;
    m[4][0]=5;
    m[4][3]=809;
    clrscr();
    printf("\n enter the input string:");
    scanf("%s",ips);
    for(i=0;ips[i];i++)
    {
        switch(ips[i])
        {
            case 'E':k=0;break;
            case 'e':k=1;break;
            case 'T':k=2;break;
            case 't':k=3;break;
            case 'F':k=4;break;
            case 'i':k=5;break;
            case '+':k=6;break;
```

```
    case '*':k=7;break;
    case '(':k=8;break;
    case ')':k=9;break;
    case '$':k=10;break;
}
ip[i]=k;
}
ip[i]=-1;
push(10);
push(0);
i=0;
printf("\tstack\t    input \n");
while(1)
{
    printf("\t");
    for(j=0;j<=top;j++)
        printf("%c",convert(stack[j]));
    printf("\t\t");
    for(k=i;ip[k]!=-1;k++)
        printf("%c",convert(ip[k]));
    printf("\n");
    if(stack[top]==ip[i])
    {
        if(ip[i]==10)
        {
            printf("\t\t SUCCESS");
            return;
        }
    }
    else
    {
        top--;
        i++;
    }
}
```

```

else if(stack[top]<=4&&stack[top]>=0)
{
    a=stack[top];
    b=ip[i]-5;
    t=m[a][b];
    top--;
    while(t>0)
        {
            push(t%10);
            t=t/10;
        }
}
else
{
    printf("ERROR");
    return;
}
}
getch();
}

```

Output:

```

enter the input string:i+i*i$
stack      input
$E        i+i*i$
$eT       i+i*i$
$eTf      i+i*i$
$eti      i+i*i$
$et       +i*i$
$e        +i*i$
$eT+      +i*i$
$eT       i*i$
$eTf      i*i$
$eti      i*i$
$et       *i$
$eTf*     *i$
$eTf      i$
$eti      i$
$et       $
$e        $
$         $
          SUCCESS_

```

Viva Questions :

1. What is top-down parsing?
2. What is parse tree?
3. How many types of parsers are there?
4. What is predictive parsing?
5. What is LL(1) grammar?

Exercise:7

Aim: Implement LALR bottom up parser for the given language.

$S \rightarrow CC$
 $C \rightarrow cC$
 $C \rightarrow d$

Description:

LALR parsing is one of the bottom-up parsing technique. In this parsing the parse table is used with input. The parse table doesn't contain multiple entries. So there is no backtracking in LALR parser. LALR parser is the efficient bottom-up parser.

Algorithm:

Input: An augmented grammar G' .

Output: The LALR parsing table actions and goto for G' .

Method:

1. Construct $C = \{I_0, I_1, I_2, \dots, I_n\}$, the collection of sets of LR(1) items.
2. For each core present in among these sets, find all sets having the core, and replace these sets by their union.
3. Parsing action table is constructed as for Canonical LR.
4. The goto table is constructed by taking the union of all sets of items having the same core. If J is the union of one or more sets of LR (1) items, that is, $J = I_1 \cup I_2 \cup \dots \cup I_k$, then the cores of $\text{goto}(I_1, X)$, $\text{goto}(I_2, X)$, ..., $\text{goto}(I_k, X)$ are the same as all of them have same core. Let K be the union of all sets of items having same core as $\text{goto}(I_1, X)$. Then $\text{goto}(J, X) = K$.

Program:

```
#include<stdio.h>
int st[20],top=-1;
char input[20];
int encode(char ch)
{
    switch(ch)
    {
        case 'c':return 0;
        case 'd':return 1;
        case '$':return 2;
        case 'S':return 3;
        case 'C':return 4;
    }
    return -1;
}
char decode(int n)
{
    switch(n)
    {
        case 0:return('c');
        case 1:return('d');
        case 2:return('$');
        case 3:return('S');
        case 4:return('C');
```

```

    }
    return 'z';
}
void push(int n)
{
    st[++top]=n;
}
int pop()
{
    return(st[top--]);
}
void display(int p,char *ptr)
{
    int l;
    for(l=0;l<=top;l++)
    {
        if(l%2==1)
            printf("%c",decode(st[l]));
        else
            printf("%d",st[l]);
    }
    printf("\t");
    for(l=p;ptr[l];l++)
        printf("%c",ptr[l]);
    printf("\n");
}
void main()
{
    char t1[20][20],pr[20][20],xy;
    int inp[20],t2[20][20],gt[20][20];
    int i,k,x,y,tx=0,ty=0,len;
    clrscr();
    strcpy(pr[1],"S-CC");
    strcpy(pr[2],"C-cC");
    strcpy(pr[3],"C-d");
    t2[0][0]=3;
    t2[0][1]=4;
    t2[2][0]=3;
    t2[2][1]=4;
    t2[3][0]=3;
    t2[3][1]=4;
    t2[4][0]=t2[4][1]=t2[4][2]=3;
    t2[5][2]=1;
    t2[6][0]=t2[6][1]=t2[6][2]=2;
    t1[0][0]=t1[0][1]='s';
    t1[1][2]='a';
    t1[2][0]=t1[2][1]='s';
    t1[3][0]=t1[3][1]='s';
    t1[4][0]=t1[4][1]=t1[4][2]='r';
}

```

```

t1[5][2]='r';
t1[6][0]=t1[6][1]=t1[6][2]='r';
gt[0][3]=1;
gt[0][4]=2;
gt[2][4]=5;
gt[3][4]=6;
printf("enter string:");
scanf("%s",input);
for(k=0;input[k];k++)
{
    inp[k]=encode(input[k]);
    if(input[k]<0||inp[k]>2)
        printf("\n error in input");
}
push(0);
i=0;
while(1)
{
    x=st[top];y=inp[i];
    display(i,input);
    if(t1[x][y]=='a')
    {
        printf("string is accepted \n");
        exit(0);
    }
    else if(t1[x][y]=='s')
    {
        push(inp[i]) ;
        push(t2[x][y]);
        i++;
    }
    else if(t1[x][y]=='r')
    {
        len=strlen(pr[t2[x][y]])-2;
        xy=pr[t2[x][y]][0];
        ty=encode(xy);
        for(k=1;k<=2*len;k++)
            pop();
        tx=st[top];
        push(ty);
        push(gt[tx][ty]);
    }
    else
    {
        printf("\n error in parsing");
        break;
    }
}
getch();
}

```

Output:

```
enter string:cdd$
0      cdd$
0c3    dd$
0c3d4  d$
0c3c6  d$
0c2    d$
0c2d4  $
0c2c5  $
0s1    $
string is accepted
```

Viva Questions:

1. What is bottom-up parsing?
2. What are the different types of LR parsers?
3. What is lookahead?
4. What is shift-reduce parsing?
5. What is LALR parsing?

Exercise:8

Aim: Implement the syntax analyzer using YACC tool.

Description:

Yacc (Yet Another Compiler-Compiler) is a computer program for the Unix operating system. An Yacc tool can be used to generate automatically an LALR parser.

A Yacc program consists of three parts:

definitions

% %

rules

% %

Subroutines

- The definition part consists of token declarations and C code bracketed by %{ and % }
- The grammar is placed in the rules section.
- User subroutines are added in subroutines section.

Program:

```
% {
#include<stdio.h>
% }
%token num
%%
E:E'\n' { printf("\n E->E:%d",$$);exit(0);}
|E'+T' {$$=$1+$3; printf("\n E->E+T:%d",$$);}
|T {$$=$1; printf("\n E->T:%d",$$);}
;
T:T'*F' {$$=$1*$3; printf("\n T->T*F:%d",$$);}
|F {$$=$1; printf("\n T->F:%d",$$);}
;
F:'(E)' {$$=$2;printf("\n F->(E):%d",$$);}
|num {$$=$1; printf("\n F->num:%d",$$);}
;
%%
main()
{
return yyparse();
}
int yylex()
{
int c;
while((c=getchar())!=' ');
if(isdigit(c))
{
ungetc(c,stdin);
scanf("%d",&yylval);
return num;
}
if(c=='\n')
{
```

```
return 0;
}
else
return c;
}
char yyerror(char *s)
{
printf("%s",s);
}
```

Output:

```
"l2.y" 42L, 627C written
[cse17_e2@localhost ~]$ yacc l2.y
[cse17_e2@localhost ~]$ cc y.tab.c
l2.y:24:22: warning: multi-character character constant
[cse17_e2@localhost ~]$ ./a.out
2+8*10

F->num:2
T->F:2
E->T:2
F->num:8
T->F:8
F->num:10
T->T*F:80
E->E+T:82[cse17_e2@localhost ~]$ █
```

Viva Questions:

1. What is YACC?
2. What is the difference between LEX and YACC?
3. What is the use of yyparse()?

Additional Experiments

Exercise:1

Aim: Write a program to take an example subnet of hosts and obtain the broadcast tree for it.

Description:

Kruskal's algorithm is used to obtain the broadcast tree from the given subnet. This algorithm constructs a minimal spanning tree for a connected weighted graph G. Algorithm :

1. Select any edge of minimal value that is not a loop. This is the first edge of T.
2. Select any remaining edge of G having minimal value that does not form a circuit with the edges already included in T.
3. Continue step-2 until T contains n-1 edges where n is the number of vertices of G.

Program:

```
#include<stdio.h>
#include<conio.h>
int n,weight[10][10]={0},intree[10]={0},d[10]={0},whoto[10]={0};
void updatedistance(int target)
{
    int i;
    for(i=0;i<n;i++)
    {
        if((weight[target][i]!=0)&&(d[i]>weight[target][i]))
        {
            d[i]=weight[target][i];
            whoto[i]=target;
        }
    }
}
void main()
{
    int i,j,total=0,s;
    printf("Enter the number of nodes : ");
    scanf("%d",&n);
    printf("Enter distance from\n");
    for(i=0;i<n;++i)
    {
        for(j=0;j<n;j++)
        {
            if(i==j)
                weight[i][j]=0;
            else
            {
                printf("%c --> %c : ", 'A'+i,'A'+j);
                scanf("%d",&weight[i][j]);
            }
        }
    }
    printf("The Distance matrix is\n");
    for(i=0;i<n;i++)
    {
```



```

        for(j=0;j<n;j++)
            printf("%d ",weight[i][j]);
        printf("\n");
    }
    for(i=0;i<n;i++)
        d[i]=1000;
    printf("Enter source from 0 to %d : ",n-1);
    scanf("%d",&s);
    printf("Broadcast tree for source node %c is\n",s+'A');
    printf("Source\tDestination\n");
    intree[s]=1;
    updatedistance(s);
    for(j=0;j<n-1;j++)
    {
        int min=-1;
        for(i=0;i<n;i++)
        {
            if(!intree[i])
            {
                if((min==-1)||(d[min]>d[i]))
                    min=i;
            }
        }
        printf("%c\t%c\n",whoto[min]+'A',min+'A');
        intree[min]=1;
        total+=d[min];
        updatedistance(min);
    }
    printf("Total distance : %d\n",total);
}

```

Output:

Test Case - 2	
User	Output
	Enter the number of nodes : 3
	Enter distance from 1
A -->	B : 1
A -->	C : 0
B -->	A : 0
B -->	C : 1
C -->	A : 1
C -->	B : 1
	The Distance matrix is 0
	0 1 0 0
	0 0 1 0
	1 1 0 0
	Enter source from 0 to 2 : 0
	Broadcast tree for source node A is
Source	Destination
A	B
B	C
	Total distance : 2

Test Case - 1
User Output
Enter the number of nodes : 5
Enter distance from 1
A --> B : 1
A --> C : 1
A --> D : 0
A --> E : 0
B --> A : 1
B --> C : 1
B --> D : 1
B --> E : 0
C --> A : 1
C --> B : 1
C --> D : 0
C --> E : 0
D --> A : 0
D --> B : 1
D --> C : 0
D --> E : 1
E --> A : 0
E --> B : 0
E --> C : 0
E --> D : 1
The Distance matrix is 2
0 1 1 0 0 2
1 0 1 1 0 2
1 1 0 0 0 2
0 1 0 0 1 2
0 0 0 1 0 2
Enter source from 0 to 4 : 2
Broadcast tree for source node C is
Source Destination
C A
C B
B D
D E
Total distance : 4

Viva Questions:

1. What is mask?
2. What is the purpose of mask?
3. What is subnet?

Exercise:2

Aim: Write a program to implement caesar cipher substitution technique.

Description:

In cryptography, a Caesar cipher, also known as Caesar's cipher, the shift cipher, Caesar's code or Caesar shift, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plain text is replaced by a letter some fixed number of positions down the alphabet. For example, with a left shift of 3, D would be replaced by A, E would become B, and so on. The method is named after Julius Caesar, who used it in his private correspondence. The transformation can be represented by aligning two alphabets; the cipher alphabet is the plain alphabet rotated left or right by some number of positions.

Program:

```
#include<stdio.h>
#include<string.h>
int main()
{
    char msg[100],ch;
    int i,key;
    printf("Enter the string : ");
    gets(msg);
    printf("Enter the key value : ");
    scanf("%d",&key);
    printf("Original string : %s\n",msg);
    for(i=0;msg[i]!='\0;++i)
    {
        ch=toupper(msg[i]);
        if(ch>='a' && ch<='z')
        {
            ch=ch+key;
            if(ch>'z')
            {
                ch=ch-'z'+'a'-1;
            }
            msg[i]=ch;
        }
        else if(ch>='A' && ch<='Z')
        {
            ch=ch+key;
            if(ch>'Z')
            {
                ch=ch-'Z'+'A'-1;
            }
            msg[i]=ch;
        }
    }
    printf("Encrypted string : %s\n",msg);
    for(i=0;msg[i]!='\0;++i)
    {
```

```

        ch=msg[i];
        if(ch>='a'&&ch<='z')
        {
            ch=ch-key;
            if(ch<'a')
            {
                ch=ch+'z'-'a'+1;
            }
            msg[i]=ch;
        }
        else if(ch>='A'&&ch<='Z')
        {
            ch=ch-key;
            if(ch<'A')
            {
                ch=ch+'Z'-'A'+1;
            }
            msg[i]=ch;
        }
    }
    printf("Decrypted string : %s\n",msg);
    return 0;
}

```

Output:

Test Case - 1
User Output
Enter the string : attackatonce
Enter the key value : 4
Original string : attackatonce
Encrypted string : EXXEGOEXSRGI
Decrypted string : ATTACKATONCE

Test Case - 2
User Output
Enter the string : abcdefghijklmnopqrstuvwxyz
Enter the key value : 23
Original string : abcdefghijklmnopqrstuvwxyz
Encrypted string : XYZABCDEFGHIJKLMNQPQRSTUVWXYZ
Decrypted string : ABCDEFGHIJKLMNQPQRSTUVWXYZ

Test Case - 3
User Output
Enter the string : axzd
Enter the key value : 4
Original string : axzd
Encrypted string : EBDH
Decrypted string : AXZD

Viva Questions:

1. What is cipher?
2. What are types of cipher?

Exercise:3

Aim: Write a program to implement rail fence cipher transposition technique.

Description:

A transposition cipher is one in which plain text symbols are rearranged (i.e., transposed or permuted) to produce cipher text. The method of transposition may be either mathematical or typographical in nature. The Rail fence cipher is a transposition cipher. It rearranges the plain text letters by drawing them in a way that they form a shape of the rails of an imaginary fence. To encrypt the message, the letters should be written in a zigzag pattern, going downwards and upwards between the levels of the top and bottom imaginary rails. The shape that is formed by the letters is similar to the shape of the top edge of the rail fence. Next, all the letters should be read off and concatenated, to produce one line of cipher text. The letters should be read in rows, usually from the top row down to the bottom one. The secret key is the number of levels in the rail. It is also a number of rows of letters that are created during encryption. This number cannot be very big, so the number of possible keys is quite limited.

Program:

```
#include<stdio.h>
#include<string.h>
void main()
{
    int n,L,i,j,k=-1,row=0,col=0;
    char a[40],b[40][40];
    printf("Enter the plain text : ");
    gets(a);
    printf("Enter the value of n : ");
    scanf("%d",&n);
    L=strlen(a);
    b[n][L];
    for(i=0;i<n;++i)
    {
        for(j=0;j<L;++j)
        {
            b[i][j]='\n';
        }
    }
    for(i=0;i<L;++i)
    {
        b[row][col++]=a[i];
        if(row==0 || row==n-1)
            k=k*(-1);
        row=row+k;
    }
    printf("The encrypted code : ");
    for(i=0;i<n;++i)
    {
        for(j=0;j<L;++j)
        {
            if(b[i][j]!='\n')
            {
```

```

        printf("%c",b[i][j]);
    }
}
printf("\nDecrypted message : ");
puts(a);
printf("\n");
}

```

Output:

Test Case - 1
User Output
Enter the plain text : defend the east wall
Enter the value of n : 3
The encrypted code : dnhaweedtees alf t1
Decrypted message : defend the east wall

Test Case - 2
User Output
Enter the plain text : attack at once
Enter the value of n : 2
The encrypted code : atc toctaka ne
Decrypted message : attack at once

Test Case - 3
User Output
Enter the plain text : This is a test
Enter the value of n : 3
The encrypted code : T ashsi etist
Decrypted message : This is a test

Viva Questions:

1. What is transposition technique?
2. What is the difference between transposition technique and substitution technique?

Exercise:4

Aim: Write a program to implement RSA algorithm to encrypt a text data and decrypt the same.

Description:

RSA is an algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of the keys can be given to anyone. The other key must be kept private. The algorithm is based on the fact that finding the factors of a large composite number is difficult: when the integers are prime numbers, the problem is called prime factorization. It is also a key pair (public and private key) generator.

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>
long int p,q,n,t,flag,e[100],d[1000],temp[100],j,m[100],en[100],i;
char msg[100];
int prime(long int);
void ce();
long int cd(long int);
void encrypt();
void decrypt();
int main()
{
    printf("Enter first prime number : ");
    scanf("%ld",&p);
    flag=prime(p);
    if(flag==0)
    {
        printf("Wrong input\n");
        exit(1);
    }
    printf("Enter another prime number : ");
    scanf("%ld",&q);
    flag=prime(q);
    if(flag==0||p==q)
    {
        printf("Wrong input\n");
        exit(1);
    }
    printf("Enter message : ");
    fflush(stdin);
    scanf("%s",msg);
    for(i=0;msg[i]!=NULL;i++)
        m[i]=msg[i];
    n=p*q;
    t=(p-1)*(q-1);
    ce();
    printf("Possible values of e and d are :");
    for(i=0;i<j-1;i++)
        printf("\n%ld\t%ld",e[i],d[i]);
}
```

```

encrypt();
decrypt();
return 0;
}
int prime(long int pr)
{
    int i;
    j=sqrt(pr);
    for(i=2;i<=j;i++)
    {
        if(pr%i==0)
            return 0;
    }
    return 1;
}
void ce()
{
    int k=0;
    for(i=2;i<t;i++)
    {
        if(t%i==0)
            continue;
        flag=prime(i);
        if(flag==1&&i!=p&&i!=q)
        {
            e[k]=i;
            flag=cd(e[k]);
            if(flag>0)
            {
                d[k]=flag;
                k++;
            }
            if(k==99)
                break;
        }
    }
}
long int cd(long int x)
{
    long int k=1;
    while(1)
    {
        k=k*t;
        if(k%x==0)
            return(k/x);
    }
}
void encrypt()
{
    long int pt,ct,key=e[0],k,len;
    i=0;
    len=strlen(msg);
    while(i!=len)
    {
        pt=m[i];
        pt-=96;
        k=1;
        for(j=0;j<key;j++)
        {
            k*=pt;
            k%=n;
        }
        temp[i]=k;
        ct=k+96;
        en[i]=ct;
        i++;
    }
    en[i]=-1;
    printf("\nThe encrypted message is : ");
    for(i=0;en[i]!=-1;i++)
        printf("%c",en[i]);
}
void decrypt()
{
    long int pt,ct,key=d[0],k;
    i=0;
    while(en[i]!=-1)
    {
        ct=temp[i];

```



```

    k=1;
    for(j=0;j<key;j++)
    {
        k*=ct;
        k%=n;
    }
    pt=k+96;
    m[i]=pt;
    i++;
}
m[i]=-1;
printf("\nThe decrypted message is : ");
for(i=0;m[i]!=-1;i++)
    printf("%c",m[i]);
printf("\n");
}

```

Output:

```

Enter first prime number : 5
Enter another prime number : 7
Enter message : theunitedkingdom
Possible values of e and d are :
11 11
13 13
17 17
The encrypted message is : tvjundtjipdn|io{
The decrypted message is : theunitedkingdom

```

```

Enter first prime number : 5
Enter another prime number : 9
Wrong input

```

Viva Questions:

1. What is encryption?
2. What is decryption?
3. What is private key?
4. What is public key?