OPERATING SYSTEM & COMPILER DESIGN LAB MANUAL

III Year I Semester

2019 - 20

Prepared by:

Sri. B.Srinivasu Kumar — **Assoc. Professor**

Smt.PNS.Sravanthi — **Assistant Professor**

GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institution with Permanent Affiliation to JNTUK, Kakinada)

Seshadri Rao Knowledge Village, Gudlavalleru – 521356

GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institution with Permanent Affiliation to JNTUK, Kakinada)

Seshadri Rao Knowledge Village, Gudlavalleru – 521356

DEPARTMENT OF INFORMATION TECHNOLOGY

INSTITUTE VISION & MISSION

Institute 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.

Institute 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&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.

DEPARTMENT VISION & MISSION

VISION

To be a centre of innovation by adopting changes in Information Technology, imparting quality education, research to produce visionary computer professionals and entrepreneurs.

MISSION

- To provide an academic environment in which students are given the essential resources for solving real-world problems and work in multidisciplinary teams.
- To impart value based education and research among students, particularly belonging to rural areas, for their sustained growth in technological aspects and leadership.
- To collaborate with the industry for making the students adoptable to evolving changes in Information Technology and related areas.

PROGRAMME EDUCATIONAL OBJECTIVES(PEOs):-

PEO1: To exhibit analytical skills in modeling and solving computing problems by applying mathematical, scientific and engineering knowledge and to pursue their higher studies.

PEO2: To communicate effectively with multi-disciplinary teams to develop quality software systems with an orientation towards research and development for lifelong learning.

PEO3: To address industry and societal needs for the growth of global economy using emerging technologies by following professional ethics.

PROGRAM OUTCOMES (POs)

Engineering students 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

Students will be able to

13.

PSO1: Organize, maintain and protect IT Infrastructural resources.

PSO2: Design and Develop web, mobile, and smart apps based software solutions to the real

Course Objectives

- To practice memory management techniques, CPU and disk scheduling algorithms.
- To implement Bankers algorithm to avoid deadlock.
- To implement lexical analyzer and syntax analyzer.

Course Outcomes

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

- implement CPU and disk scheduling algorithms.
- develop code for memory management techniques.
- design codeto implement Bankers algorithm to avoid dead locks.
- implement lexical analyzer and syntax analyzer

OPERATING SYSTEMS AND COMPILER DESIGN LAB	1	2	3	4	5	6	7	8	9	10	11	12	PSO 1	PSO 2
CO1. implement CPU and disk scheduling algorithms.	3	2	2					2	2	2			2	
CO2: develop code for memory management techniques.	2	2	2					2	2	2			2	
CO3: design code to implement Bankers algorithm to avoid dead locks.	2	2	2					2	2	2			2	
CO4: implement lexical analyzer and syntax analyzer.	2	2	2					2	2	2			2	
	2	2	2					2	2	2			2	

S.No	EXPERIMENTS	Page No
	OPERATING SYSTEMS	
1(a)	To simulate the CPU scheduling algorithm using Round-Robin methodology .(Round Robin)	
1(b)	To simulate the CPU scheduling algorithm using Shortest Job First methodology (SJF)	
1(c)	To simulate the CPU scheduling algorithm using First Come First Serve methodology (FCFS)	
1(d)	To simulate the CPU scheduling algorithm using Priority methodology (Priority)	
2	To implement and simulate the MVT and MFT techniques.	
3(a)	To implement page replacement algorithm using FIFO mechanism.	
3(b)	To implement the algorithm for the page replacement algorithm using LRU mechanism.	
3(c)	To implement the algorithm for the page replacement algorithm using LFU mechanism.	
4	To simulate Bankers algorithm for Deadlock Avoidance.	
5	To implement Disk Scheduling Algorithm using A) FCFS B) SSTF technique	
	COMPILER DESIGN	
6	Design lexical analyzer to recognize the tokens and removes the comment lines and blank spaces.	
7	Implement the lexical analyzer using JLEX/ FLEX/ other lexical analyzer generating tools.	
8	a) Write a program to compute FIRST sets for the given grammar.	
	b) Write a program to compute FOLLOW sets for the given grammar. **Note**: inputs are FIRST sets and grammar.	
9	a) Implement predictive parsing table for the given grammar.	
10	Implement the syntax analyzer using YACC tool.	
	ADDITIONAL EXPERIMENT	
1	To develop a Recursive Descent Parser for the given language.	

OPERATING SYSTEMS

1. Write a program to simulate the CPU scheduling algorithm using
   a) Round-Robin methodology.
   b) Shortest Job First methodology
   c) First Come First Serve methodology
   d) Priority methodology

A) AIM: To write a program to implement CPU Scheduling using Round-Robin methodology.

PROGRAM:

```
#include<stdio.h>

main( )

{

 int a[10],b[10],w[10],n,i,j,k,tq,et;

 float avg=0,tt=0;

 printf("\n enter no. of processes");        /* reading no of processes */

scanf("%d",7n);

printf("enter burst times");                 /* reading burst times */

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

{

scanf("%d",&a[i]);

if(a[i]<0)

{

      printf("\n INVALID INPUT");

      exit(0);

}

tt=tt+a[i];

b[i]=a[i];

w[i]=0;

}
```

```
printf(“enter time quantum”);
    /* reading time quantum */
    scanf(“%d”,&tq);
    i=0;
    while(i<n)
    {
    if(a[i]>0)
    {
        if(a[i]>tq)
        {
        a[i]=a[i]-tq;
        et=tq;
         }
         else
        {
          et=a[i];
          a[i]=0;
         }
        for(k=0;k<n;k++)
        {
         if( (k!=i) && a[k]!=0)
        w[k]=w[k]+et;
         }
    }
   i++;

   if(i==n)
   {
```

```
    for(j=0;j<n;j++)
        {
            if(a[j]!=0)
            i=0;
                }
             }
        }
            for(k=0;k<nk++)                    /* calculating turn around time */
            avg=avg+w[k];
            tt=tt+avg;
            tt=tt/n;
            printf("\n Turnaround time: %f", tt);
            avg=avg/n;                         /* calculating average time */
            printf("the average waiting time:", avg);
            getch( );
}
```

INPUT/OUTPUT:

1) enter the time slot 4

   enter the number of processes 3

   enter the burst times 24 3 3

   turn around time is 15.66

   the average waiting time : 5.6675

2) enter the time slot 100

   enter the number of processes 3

   enter the burst time 1000 1212 17

   turn around time :1386.76

   the average turn around time is 711.3

B) AIM: To write a program to implement CPU Scheduling using Shortest Job First methodology.

PROGRAM:

```
#include<stdio.h>
main()
{
int a[10],b[10],w[10],at[10],n,s=0,i,j,k,t,t1,t2;
float avg=0,tt=0;
printf("enter no. of processes\n");              /* reading no of processes */
scanf("%d", &n);
printf("enter burst times of processes\n");   /* reading burst times for  processes */
for(i=0;i<n;i++)
{
scanf("%d", &a[i]);
if(a[i]<0)
{
 printf("invalid input");
 exit(0);
            }
 b[i]=0;
 tt=tt+a[i];

 }
 printf("enter arrival times\n");              /* reading arrival times */
 for(i=0;i<n;i++)
 scanf("%d", &at[i]);
 k=0;
 j=1;
```

```
for(i=0;i<n-1;i++)
            {
             if(a[i]!=0)
             {
             while(at[i+1]!=s)
             {
               k=i;
               for(j=0;j<i; j++)
                {
                 if(a[i]>a[j])
                 k=j;
                }
             a[k]=a[k]-1;
             b[k]++;
             s=s+1;
            }
            }
            }
               for(i=0;i<n; i++)
            {

             for(j=i; j<n; j++)
             {
               if(a[i]>a[j])   /* swapping */
            {
               t=a[i];
               a[i]=a[j];
               a[j]=t;
```

```
            t1=at[i];
            at[i]=at[j];

            at[j]=t1;
            t2=b[i];
            b[i]=b[j];
            b[j]=t2;
        }
    }
}
        for(i=0;i<n; i++)
        {
        w[i]=s-b[i];
        s=s+a[i];
        }
        for(i=0;i<n; i++)
        {
        w[i]=w[i]-at[i];
        avg=avg+w[i];
        }
        tt=tt+avg;          /* calculating average time and turnaround time */
        tt=tt/n;
        avg=avg/n;
        printf("turn around time is %f", tt);
        printf("avg waiting time is %f", avg);
        getch();
    }
```

INPUT/OUTPUT:

1) enter the number of processes 4
   enter the burst times 8 4 9 5

   enter the arrival times 0 1 2 3

   turnaround time is 13

   avg waiting time is 6.5

2) enter the number of processes 4
   enter the burst times 130 540 120 60

   enter the arrival times of processes 0 5 9 15

   turnaround time is 148

   avg waiting time is 136.25

c) AIM: To write a program to implement CPU Scheduling using FCFS methodology.

Program:

```
#include<stdio.h>
main()
{
int a[10],b[10],n, i;
float avg=0,tt=0;
printf("enter the no of processes");      /* reading no of processes */
scanf("%d", &n);
printf("\n enter the burst times");       /* reading burst times */
for(i=0;i<n; i++)
{
  scanf("%d", &b[i]);
  if(b[i]<0)
```

```
{
   printf("\n invalid input");
   exit(0);
   }
   tt=tt+b[i];                    /* calculating turnaround time */
   }
     a[0]=0;
for(i=1;i<n; i++)
   {
   a[i]=a[i-1]+b[i-1];
   avg=avg+a[i];
   }
  tt=tt+avg;
  tt/n;
  avg/n;
   printf("Turn around time:%f", tt);
  printf("\n Avg. Waiting time:%f", avg);
  getch();
 }
```

Input/output:

1) enter the no of processes 3
   enter the burst time

   24 3 3

   The average waiting time is:17.0

   The turn around time is:27.0

2) enter the no of processes 6

   enter the burst time -5

   Invalid input

d) AIM: To write a program to implement CPU Scheduling using Priority methodology.

PROGRAM:

```
#include<stdio.h>
main()
{
  int bt[10],w[10],p[10],I,j,n,s,t,t1;


  float avg=0,tt=0.
  clrscr();
  printf("enter no. of processes");              /* reading processes */
  scanf("%d", &n);
  printf("enter the burst times");                * reading burst times */
  for(i=0;i<n; i++)
  {
  scanf("%d",  &bt[i]);
  if(bt[i]<0)
  {
  printf("\n INVALID DATA");
  exit(0);
 }
tt=tt+bt[i];
}
  printf("enter the priorities");                 /* reading priorities */
  for(i=0;i<n; i++)
  scanf("%d", &p[i]);
 for(i=0;i<n; i++)
```

```
    {
  for(j=i;j<n; j++)
   {
      if(p[i]>p[j])
      {
      t=bt[i];              /* swapping */
      bt[i]=bt[j];
      bt[j]=t;
      t1=p[i];
     p[i]=p[j];
    p[j]=t1;
   }
 }
}
    w[0]=0;
    s=0;
   for(i=1;i<n; i++)
   {
    s=s+bt[i-1];
    w[i]=s;
    avg=avg+w[i];
   }
```

```
    tt=tt+avg;                    /* calculating turnaround time and average time */
     tt=tt/n;
     avg=avg/n;
     printf("\n turn around time is :%f", tt);
     printf("\n average waiting time is :%f", avg);
     getch();
}
```

INPUT/OUTPUT:

```
 enter number of process 4
enter the burst time
        12 7 72 18
Enter the priorities
       3 4  1  2
Turn around time  95.0


Average waiting time   66.0
enter number of process 4
enter the burst time
       100  240  370  800
Enter the priorities
       700  800  300  100
Turn around time  1350.0
Average waiting time   810
```

2) Write a Program to Simulate the MVT and MFT.

a) AIM: A program to simulate the MVT (Multiprogramming Using Variable Number of Tasks).

PROGRAM:

```
#include<
stdio.h>
#include<
conio.h>
void
main()
{
int m=0, m1=0,m2=0,p,
count=0,i; clrscr();
printf("enter the memory
capacity:"); scanf("%d",
&m);
printf("enter the no of
processes:"); scanf("%d",
&p);
for(i=0;i<p; i++)
{
printf("\n enter memory req for
process%d:",i+1);
 scanf("%d",&m1);
count=co
unt+m1;
if(m1<=
m)
{

if(count==m)
printf("there is no further memory remaining:");
printf("the memory allocated for process%d is: %d
",i+1,m);
m2=m-m1;
printf("\nremaining memory
is:%d",m2);
m=m2;
}
}
else



{
printf("memory is not allocated for process%d",i+1);
}
```

```
printf("\n external fragmentation for this process is:%d", m2);
}
getch();
```

INPUT/OUTPUT:

Input:

Enter the memory capacity: 80

Enter no of processes: 2
Enter memory req for process1: 23

Output:

The memory allocated for process1 is: 80
Remaining memory is: 57
External fragmentation for this process is: 57
Enter memory req for process2: 52
The memory allocated for process2 is: 57
Remaining memory is: 5
External fragmentation for this process is: 5

b)MFT:

AIM: A Program to simulate the MFT (Multiprogramming with fixed number of tasks)

PROGRAM:

```
#include<stdio.h>
#include<conio.h>          int
main()
                {
                int m, p,s,p1;
                int
                m1[4],i,f,f1=0,f2=0,fra1,fra2,s1,
                pos; clrscr();



                printf("Enter          the
                memorysize:"):
                scanf("%d",&m);
                printf("Enter      the      no
                ofpartitions:");
                scanf("%d",&p);
                s=m/p;
                printf("Each partition size
                is:%d",s); printf("\nEnter the
                no              ofprocesses:");
                scanf("%d",&p1);
                pos=m;
                for(i=0;i
                <p1;i++)
                {
                if(pos<s)
                {
                printf("\nThere is no further memory for
                process%d",i+1); m1[i]=0;
                break;
                }
                else
                {
                printf("\nEnter the memory req for



                process%d:",i+1);
                scanf("%d",&m1[i]);



                if(m1[i]<=s)
                {
                printf("\nProcess          is          allocated
```

```
inpartition%d",i+1);
fra1=s-m1[i];
printf("\nInternal fragmentation for
processis:%d",fra1);
f1=f1+fra1;
pos=pos-s;
}
else
{
printf("\nProcess not allocated in
partition%d",i+1); s1=m1 [i];

while(s1>s)
{
s1=s1-s; pos=pos-s;
}
pos=pos-s; fra2=s-s1; f2=f2+fra2;
printf("\nExternal Fragmentation for this process is:%d",fra2);
}
}
}
printf("\nProcess\tallocatedm
emory"); for(i=0; i<p1; i++)
printf("\n%5d\t%5d",i+1,
m1[i]); f=f1+f2;



printf("\nThe tot no of
fragmentationis:%d",f);
getch();
return 0;}
```

INPUT/OUTPUT:

Input:

Enter the memory size: 80
Enter the no of partitions: 4
Each partition size: 20
Enter the number of processes: 2
Enter the memory req for process1: 18

Output:

Process1 is allocated in partn1
Internal fragmentation for process1 is: 2
Enter the memory req for process2: 22
Process2 is not allocated in partn2
External fragmentation for process2 is: 18

Process	memory	allocated
1	20	18
2	20	22

The tot no of fragmentation is: 20

3) Write a Program Simulate all Page Replacement Algorithms

a) **FIFO**
b) **LRU**
c) **LFU**

a) FIFO:

AIM: To simulate FIFO Page Replacement Algorithm

PROGRAM:

```
#include<
stdio.h>
#include<
conio.h>
void
main()
{
int
a[5],b[20],n,p=0,q=0,m=0,h,k,i,q
1=1; char f='F';
clrscr();
printf("Enter the Number of
Pages:");
scanf("%d", &n);
printf("Enter %d
PageNumbers:",n);
for(i=0;i<n; i++)
scanf("%d
", &b[i]);
for(i=0;i<n
; i++)
{if(p==0)
{
if(q>=3) q=0; a[q]=b[i]; q++; if(q1<3)
{
q1=q;
}
}
printf("\n%d", b[i]); printf("\t"); for(h=0;h<q1; h++) printf("%d", a[h]);
if((p==0)&&(q<=3))
{
printf("-->%c",f);
m++;
}
p=0;
 for(k=0;k<q1;k++)
```

```
{
if(b[i+1]==a[k])
p=1;
}
}
printf("\nNo offaults:%d",m);
getch();
}
```

INPUT/OUTPUT:

Input:

Enter the Number of Pages: 12
Enter 12 Page Numbers:
2 3 2 1 5 2 4 5 3 2 5 2

Output:

2 2-> F
3 23-> F
2 23
1 231-> F
5 531-> F
2 521-> F
4 524-> F
5 524
3 324-> F
2 324
5 354-> F
2 352-> F

No of faults: 9

b) LRU:

AIM: To simulate LRU Page Replacement Algorithm

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int g=0,a[5],b[20],p=0,q=0,m=0,h,k,i,q1=1,j,u,n;
char f='F';
```

```
clrscr();
printf("Enter the number of pages:");
scanf("%d", &n);
printf("Enter %d Page Numbers:",n);
for(i=0;i<n; i++)
scanf("%d", &b[i]);
for(i=0;i<n; i++)
{if(p==0)
{
if(q>=3)
q=0;
a[q]=b[i];
q++;
if(q1<3)
{
q1=q;
}
}
printf("\n%d", b[i]);
printf("\t");
for(h=0;h<q1;h++)
printf("%d", a[h]);
if((p==0)&&(q<=3))
{
printf("-->%c", f);
m++;
}
p=0;
g=0;
if(q1==3)
{
for(k=0;k<q1;k++)
{
if(b[i+1]==a[k])
p=1;
}
for(j=0;j<q1;j++)
{
u=0;
k=i;
while(k>=(i-1)&&(k>=0))
{
if(b[k]==a[j])
u++;
k--;
}
if(u==0)
```

```
q=j;
}
}
else
{
for(k=0;k<q; k++)
{
if(b[i+1]==a[k])
p=1;
}
}
}
printf("\nNo of faults:%d", m);
getch();
}
```

INPUT/OUTPUT:

Input:

Enter the Number of Pages: 12
Enter 12 Page Numbers:
2 3 2 1 5 2 4 5 3 2 5 2

Output:

2	2-> F
3	23-> F
2	23
1	231-> F
5	251-> F
2	251
4	254-> F
5	254
3	354-> F
2	352-> F
5	352
2	352

No of faults: 7

4) Write a Program to Simulate Banker's Algorithm for Deadlock Avoidance.

AIM: To simulate the Bankers Algorithm for Deadlock Avoidance.

PROGRAM:

```
//Bankers algorithm for deadlock
avoidance. #include<stdio.h>
#include
<conio.h
>     void
main()
{
int n, r, i, j, k, p, u=0,s=0,m;
int                              block[10],
un[10],active[10],newreq[10];        int
max[10][10],          resalloc[10][10],
resreq[10][10];                      int
totalloc[10],totext[10],simalloc[10];
clrscr();
printf("Enter the no of
processes:");
scanf("%d",&n);
printf("Enter the no of resource
classes:"); scanf("%d",&r);
printf("Enter the total existed resource in
eachclass:");
for(k=1;k<=r;k++)
scanf("%d",&totext[k]);
printf("Enter                the
allocatedresources:");
for(i=1;i<=n;i++)
for(k=1;k<=r;
k++)
scanf("%d",&r
esalloc);
printf("Enter the process making the
newrequest:");
scanf("%d",&p);
printf("Enter                the
requestedresource:");
for(k=1;k<=r;k++)
scanf("%d",&newreq[k]);
printf("Enter the process which are n blocked
orrunning:");
for(i=1;i<=n;i++)
{
if(i!=p)
```

```
{
printf("process
%d:\n",i+1);
scanf("%d%d",&block[i],
&run[i]);
}
}
block[p]=
0;
run[p]=0;
for(k=1;k
<=r;k++)
{
j=0;
for(i=1;i
<=n;i++
)
{



totalloc[k]=j+resa
lloc[i][k];
j=totalloc[k];
}
}
for(i=1;i<=n;i++)
{
if(block[i]==1||r
un[i]==1)
active[i]=1;
else active[i]=0;
}
for(k=1;k<=r;k++)
{
resalloc[p][k]+=ne
wreq[k];
totalloc[k]+=newr
eq[k];
}
for(k=1;k<=r;k++)
{
if(totext[k]-totalloc[k]<0)
{
u=1;break;
}
}
if(u==0)
{
```

```
for(k=1;k<=r;
k++)
simalloc[k]=to
talloc[k];
for(s=1;s<=n;s
++)
for(i=1;i<=n;i
++)
{
if(active[i]==1)
{
j=0;
for(k=1;k
<=r;k++)
{
if ((totext[k]-simalloc[k])<(max[i][k]-resalloc[i][k]))
{
j=1;break;
}
}
}
if(j==0)
active[i]=0;
for(k=1;k<=r;k++
)
simalloc[k]=resal
loc[i][k];
}
}
m=0;



for(k=1;k<=r;k
++)
resreq[p][k]=n
ewreq[k];
printf("Deadlock willn't occur");
}
else
{
for(k=1;k<=r;k++)
{
resalloc[p][k]=n
ewreq[k];
totalloc[k]=newr
eq[k];
}
printf("Deadlock will occur");
}
getch();
}
```

INPUT/OUTPUT:

Input:

Enter the no of resources: 4
Enter the no of resource classes: 3
Enter the total existed resources in each class: 3 2 2
Enter the allocated resources: 1 0 0 5 1 1 2 1 1 0 0 2
Enter the process making the new request: 2

Enter the requested resource: 1 1 2
Enter the processes which are n blocked or running:
Process 1: 1 2
Process 3: 1 0
Process 4: 1 0

Output:

Deadlock will occur

5. Write a) Program to Implement Disk Scheduling using FCFS approach

Aim: To Implement Disk Scheduling using FCFS approach

Program:

```
#include<stdio.h>
#include<stdio.h>
#include<conio.h>
void main()
{
int a[10], i, n, dpos, sum=0;
clrscr();
printf("\n enter the number of requests:");
scanf("%d", &n);
printf("\n enter the disk queue:");
for(i=0; i<n; i++)
{
scanf("%d", &a[i]);
printf("\n");
}
printf("\n enter the initial head disk position:");
scanf("%d", &dpos);
for(i=0;i<n; i++)
{
sum=sum+abs(a[i]-dpos);
dpos=a[i];
}
printf("\n total head movement:%5d cylinders", sum);
getch();
}
```

INPUT / OUTPUT:

```
 enter the number of requests:6
 enter the disk queue:11

22

44

55

66

77
enter the initial head disk position:11

total head movement:   66 cylinders
```

*

5. Write programs to implement the following disk scheduling algorithms
b) SSTF
SSTF Disk Scheduling Algorithm

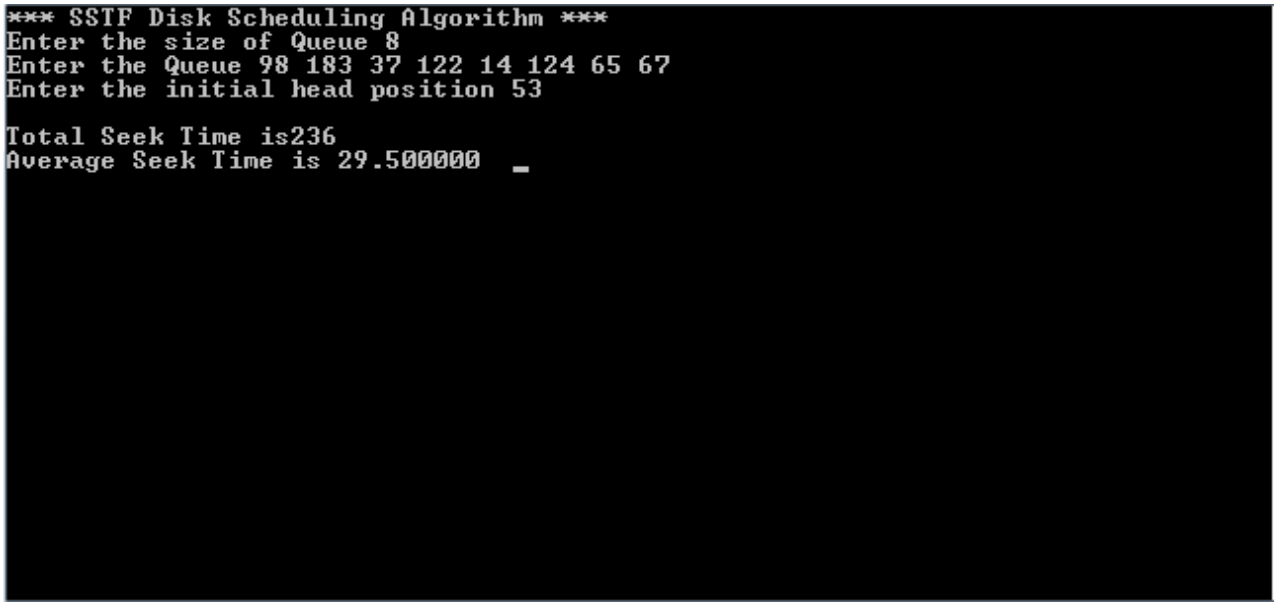
Aim: To implement the SSTF disk scheduling algorithms.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
        int queue[100],t[100],head,seek=0,n,i,j,temp;
        float avg;
      // clrscr();
        printf("*** SSTF Disk Scheduling Algorithm ***\n");
        printf("Enter the size of Queue\t");
        scanf("%d",&n);
        printf("Enter the Queue\t");
        for(i=0;i<n;i++)
        {
                scanf("%d",&queue[i]);
        }
        printf("Enter the initial head position\t");
        scanf("%d",&head);
        for(i=1;i<n;i++)
        t[i]=abs(head-queue[i]);
        for(i=0;i<n;i++)
        {
                for(j=i+1;j<n;j++)
                {
                        if(t[i]>t[j])
                        {
                                temp=t[i];
                                t[i]=t[j];
                                t[j]=temp;
                                temp=queue[i];
                                queue[i]=queue[j];
                                queue[j]=temp;
                        }
                }
        }
        for(i=1;i<n-1;i++)
        {
                seek=seek+abs(head-queue[i]);
                head=queue[i];
        }
```

```
            printf("\nTotal Seek Time is%d\t",seek);
            avg=seek/(float)n;
            printf("\nAverage Seek Time is %f\t",avg);
            getch();
        }
```

OUTPUT:



Compiler Design:

Program 7:

```
%{
int count = 0;
%}


%%
[A-Z] {printf("%s capital letter\n", yytext);
       count++;}
.     {printf("%s not a capital letter\n", yytext);}
\n    {return 0;}
%%


int yywrap(){}
int main(){

yylex();
printf("\nNumber of Captial letters "
      "in the given input - %d\n", count);

return 0;
}
```

Compilation:

```
~
~
"prog1.l" [New] 21L, 314C written
[it18_1201@localhost ~]$ lex prog1.l
[it18_1201@localhost ~]$ gcc lex.yy.c
[it18_1201@localhost ~]$ ./a.out
cfg125sjfh
c not a capital letter
f not a capital letter
g not a capital letter
1 not a capital letter
2 not a capital letter
5 not a capital letter
s not a capital letter
j not a capital letter
f not a capital letter
h not a capital letter

Number of Captial letters in the given input - 0
[it18_1201@localhost ~]$
```

Program 2:

```
%{
int no_of_lines = 0;
int no_of_chars = 0;
%}


%%
\n      ++no_of_lines;
.       ++no_of_chars;
end     return 0;
%%

int yywrap(){}
int main(int argc, char **argv)
{

yylex();
printf("number of lines = %d, number of chars = %d\n",
       no_of_lines, no_of_chars );

return 0;
}
```

```
"prog2.l" [New] 22L, 304C written
[it18_1201@localhost ~]$ lex prog2.l
[it18_1201@localhost ~]$ gcc lex.yy.c
[it18_1201@localhost ~]$ ./a.out
my name is rama
place is hyd
rank is 12548
.....
end
number of lines = 4, number of chars = 45
[it18_1201@localhost ~]$
```

10. to implement calculator by using YACC tool

program name: calc.lex

```
%{
#include <stdio.h>
#include "y.tab.h"
int c;
extern  int  yylval;
%}
%%
" "      ;
[a-z]    {
          c = yytext[0];
          yylval = c - 'a';
          return(LETTER);
         }
[0-9]    {
          c = yytext[0];
          yylval = c - '0';
          return(DIGIT);
         }
[^a-z0-9\b]    {
                c = yytext[0];
                return(c);
               }
```

program name: calc.ycc

```
%{
#include <stdio.h>
int regs[26];
int base;
%}
%start list
%token DIGIT LETTER
%left  '|'
%left  '&'
%left  '+'  '-'
%left  '*'  '/'  '%'
%left UMINUS  /*supplies precedence for unary minus */
%%                /* beginning of rules section */
list:                 /*empty */
        |
       list stat '\n'
        |
       list error '\n'
        {
          yyerrok;
        }
        ;
stat:    expr
        {
          printf("%d\n",$1);
        }
        |
        LETTER '=' expr
        {
```

```
        regs[$1] = $3;
      }
      ;
expr:   '(' expr ')'
      {
       $$ = $2;
      }
      |
      expr '*' expr
      {
       $$ = $1 * $3;
      }
      |
      expr '/' expr
      {
       $$ = $1 / $3;
      }
      |
      expr '%' expr
      {
       $$ = $1 % $3;
      }
      |
      expr '+' expr
      {
       $$ = $1 + $3;
      }
       |
      expr '-' expr
```

```
        {
          $$ = $1 - $3;
        }
        |
        expr '&' expr
        {
          $$ = $1 & $3;
        }
        |
        expr '|' expr
        {
          $$ = $1 | $3;
        }
        |
       '-' expr %prec UMINUS
        {
          $$ = -$2;
        }
        |
        LETTER
        {
          $$ = regs[$1];
        }
        |
        number
        ;
number:  DIGIT
        {
          $$ = $1;
```

```
        base = ($1==0) ? 8 : 10;
      }     |
      number DIGIT
      {
        $$ = base * $1 + $2;
      }
      ;
%%
main()
{
 return(yyparse());
}
yyerror(s)
char *s;
{
  fprintf(stderr, "%s\n",s);
}
yywrap()
{
  return(1);
}
```

output:

procedure for out is:

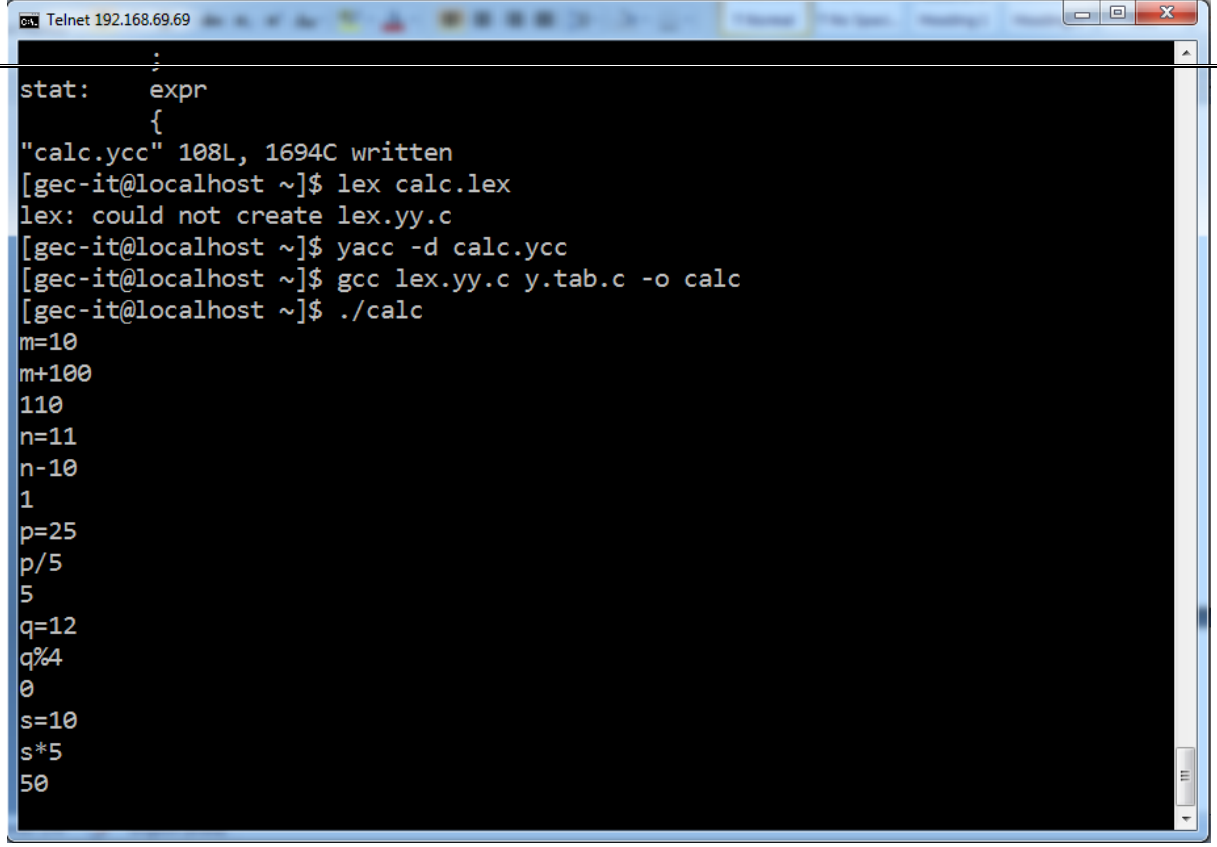
lex calc.lex

yacc -d calc.ycc

gcc lex.yy.c y.tab.c -o calc

./calc

we will get the out below


Telnet 192.168.69.69
;
stat: expr
{
"calc.ycc" 108L, 1694C written
[gec-it@localhost ~]$ lex calc.lex
lex: could not create lex.yy.c
[gec-it@localhost ~]$ yacc -d calc.ycc
[gec-it@localhost ~]$ gcc lex.yy.c y.tab.c -o calc
[gec-it@localhost ~]$ ./calc
m=10
m+100
110
n=11
n-10
1
p=25
p/5
5
q=12
q%4
0
s=10
s*5
50

EXPERIMENT-8

OBJECTIVE:

Read the input string.

By using the FIRST AND FOLLOW values.

Verify the FIRST of non terminal and insert the production in the FIRST value

If we have any @ terms in FIRST then insert the productions in FOLLOW values Constructing the predictive parser table

PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program.

PROGRAM:

```
#include<stdio.h> #include<conio.h> #include<string.h>
char prol[7][10]={"S","A","A","B","B","C","C"};

char pror[7][10]={"A","Bb","Cd","aB","@","Cc","@"};
```

```
char  prod[7][10]={"S->A","A->Bb","A->Cd","B->aB","B->@","C->Cc","C->@"}; char
first[7][10]={"abcd","ab","cd","a@","@","c@","@"};     char
follow[7][10]={"$","$","$","a$","b$","c$","d$"}; char table[5][6][10];
numr(char c)

{

switch(c)

{

case 'S': return 0; case 'A': return 1; case 'B': return 2; case 'C': return 3; case 'a': return 0;
case 'b': return 1; case 'c': return 2; case 'd': return 3; case '$': return 4;
}
return(2);

}

void main()

{

int i,j,k;
```

```
clrscr(); for(i=0;i<5;i++) for(j=0;j<6;j++) strcpy(table[i][j]," ");
printf("\nThe following is the predictive parsing table for the following
grammar:\n"); for(i=0;i<7;i++)

printf("%s\n",prod[i]); printf("\nPredictive parsing table is\n"); fflush(stdin);
for(i=0;i<7;i++)

{

k=strlen(first[i]); for(j=0;j<10;j++)
if(first[i][j]!='@') strcpy(table[numr(prol[i][0])+1][numr(first[i][j])+1],prod[i];
}

for(i=0;i<7;i++)

{

if(strlen(pror[i])==1)

{

if(pror[i][0]=='@')

{

k=strlen(follow[i]); for(j=0;j<k;j++)
```

```
strcpy(table[numr(prol[i][0])+1][numr(follow[i][j])+1],prod[i]);
}

}

strcpy(table[0][0]," ");

strcpy(table[0][1],"a");

strcpy(table[0][2],"b");

strcpy(table[0][3],"c");

strcpy(table[0][4],"d");

strcpy(table[0][5],"$");

strcpy(table[1][0],"S");

strcpy(table[2][0],"A");

strcpy(table[3][0],"B");

strcpy(table[4][0],"C");

printf("\n   \n");
```

```
for(i=0;i<5;i++) for(j=0;j<6;j++)
{

printf("%-10s",table[i][j]); if(j==5)
printf("\n   \n");

}

getch();

}
```

INPUT & OUTPUT:

The following is the predictive parsing table for the following grammar: S->A

A->Bb A->Cd B->aB B->@ C->Cc C->@

Predictive parsing table is

----

a b c d $

---------- S S->AS->AS->AS->A

---------- A A->Bb A->BbA->Cd A->Cd

B B->aB B->@ B->@ B->@

---------- **C C->@C->@ C->@**

---

1. Write a Program for designing a Lexical Analyzer in C Language.

Aim: To develop a Lexical analyzer to recognize a few patterns in C

Program:

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
#define NUM 256
int lineno=0;
#define key 0
#define id 1
#define oper 2
#define NONE -1
#define ltrl -2
struct
{
char token[10];
int toktype;
} symbol_table[50];
int loc=0;        /* Symbol table last entry */
int kcnt=23;      /* Keywords count */
char keylist[][10]={"if","then","else","while","do","return","continue",
                    "break","for","switch","case","int","float","char",
                    "double","extern","auto","register","static","sizeof",
                     "union","main"};          /* keywords list */
```

```
main()
{
 int lv=0;
 clrscr();
 while(1)
  {
    lv=lexan();
    if(lv==-999)
    break;
    printf("---%d ", lv);
  }
  print_symbol_table();
  printf("\n No of lines = %d", lineno);
  getch();
}
lexan()
{
char lexbuf[100],c;
int tokenval, i, p;
while(1)
{
c=getchar();
if(c==EOF)
return(-999);
if (c ==' ' || c=='\t')
else if (c=='\n')
```

```
            {
            printf("\n");
            lineno++;
             }
            else if(c=='/')     /******** checking for comment *******/
            {
            c=getchar();
            if(c=='*')
            {
            while(1)
            {
            c=getchar();
            if(c=='*')
            {
            while(c=='*')
            c=getchar();
            if(c=='/')
            {
            tokenval=-1;
            printf("comment");
            return tokenval;
            }
          }
        }
      }
}
```

```
else if(c=="")    /********* checking for literal  **********/
{
  while(1)
  {
    c=getchar();
    if(c=='\"')
    {
     while(c=='\"')
     c=getchar();
     if(c!='\"')
    {
    ungetc(c,stdin);
    printf("ltrl");
    return ltrl;
      }
    }
  }
 }
else if (isdigit(c))
    {
      tokenval = c-'0';
      c=getchar();
      while(isdigit(c)) {
      tokenval = tokenval*10 + c-'0';
      c=getchar();
      }
```

```
 ungetc(c,stdin);
 return NUM;
 }
else if (isalpha(c))
{
i=0;
lexbuf[i++]=c;
c=getchar();
while(isalnum(c))
{
   lexbuf[i++]=c;
   c=getchar();
}
lexbuf[i]='\0';
ungetc(c,stdin);
for(i=0;i<kcnt;i++)
 if(equals(lexbuf, keylist[i])) break;
if(i<kcnt)
{
   printf("<%s,%s>",lexbuf,"keyword");
   return key;
}
else
{
```

```
            printf("<%s,%s>",lexbuf,"id");
            p=lookup(lexbuf);
            tokenval=id;
            if(p== -1)
            p=insert(lexbuf,tokenval);
            tokenval = p;
            return tokenval;
            }
        }
        else {
                switch(c)
                {
                    case '+' :
                    case '-' :
                    case '*' :
                    case '/' :
                    case '=' :
                            printf("<%c,op>",c);
                }
                tokenval=c;
                return tokenval;
               }
         }
    }
```

```
/************** Checking keyword *********************/
int equals(char *a,char *b)
{
 int i;
 for(i=0;a[i]==b[i] && a[i];i++);
 if(a[i]=='\0' && b[i]=='\0') return 1;
else  return 0;
}
/*************** Symbol Table Management ***************/
 /* Inserting in to Symbol Table */
 int insert(char *lexiom,int type)
{
 strcpy(symbol_table[loc].token,lexiom);
 symbol_table[loc].toktype=type;
return(loc++);
}
/* Searching for a Symbol */

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

```
/*Symbol Table entries*/
print_symbol_table()
{
 int i;
 printf("\n\nloc - symbol - type(1-ID)\n");
 for(i=0;i<loc;i++)
 printf("\n%d - %s - %d",i,symbol_table[i].token,
 symbol_table[i].toktype);
}
```

INPUT/OUTPUT:

```
main(){
<main,keyword>---0 ---40 ---41 ---123
/****** This is a  ***** comment ****/
comment----1
floa a;
<float,keyword>---0 <a,id>---0 ---59
int b;
<int,keyword>---0 <b,id>---1 ---59
float c;
<float,keyword>---0 <c,id>---2 ---59
c=a+b*c
<c,id>---2 <=,op>---61 <a,id>---0 <+,op>---43 <b,id>---1
<*,op>---42 <c,id>---2 ---59
}
```

---125

^Z

loc - symbol - type(1-ID)

0 - a - 1

1 - b - 1

2 - c - 1

No of lines = 7

2. Write a Program to implement the Lexical Analyzer using Flex tool.

Aim: To implement the Lexical Analyzer using Flex tool.

Program:

```
%{
/*  declaration part */
#include<stdio.h>
 %}
 %%
 /*  definition part */
 “int” printf(“KEYWORD”);
 “char” printf(“KEYWORD”);
  [a-zA-Z][a-zA-Z0-9]* printf(“IDENTIFIER”);
  [0-9]+ printf(“NUMBER”);
  [+*/%-=] printf(“OPERATOR”);
   “/*”[A-Za-z0-9 ]*”*/” printf(“COMMENT”);
  %%
```

Input/output:

```
$lex ex.l
```

$cc lex.yy.c –o ex –ll

$./ex

int

KEYWORD

ab12

IDENTIFIER

786876

NUMBER

/* TESTING THE COMMENT */

COMMENT

3. Write a Program to design a Predictive Parser in C Language

Aim: To design a Predictive Parser in C Language

Program:

```
/*program to implement PREDICTIVE PARSER */
#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();
```

}

INPUT/OUTPUT:

```
enter the string:i+(i*i)$
stack          input
$E             i+(i*i)$
$eT            i+(i*i)$
$etF           i+(i*i)$
$eti           i+(i*i)$
$et            +(i*i)$
$e             +(i*i)$
$eT+           +(i*i)$
$eT             (i*i)$
$etF            (i*i)$
$et)E(          (i*i)$
$et)E            i*i)$
$et)eT           i*i)$
$et)etF          i*i)$
$et)eti          i*i)$
$et)et           *i)$
$et)etF*          *i)$
$et)etF            i)$
$et)eti            i)$
$et)et             )$
$et)e              )$
$et)               )$
$et                 $
$e                  $
$                   $
```

SUCCESS

4. Write a Program to design LALR bottom up parser for the given language.

Aim: To design LALR bottom up parser for the given language.

Program:

```
/*program to illustrate LALR PARSER*/
#include<stdio.h>
int st[20],top=-1;
char input[20];
int encode(char ch)
{
  switch(ch)
```

{

```
      case 'i':return 0;
      case '+':return 1;
      case '*':return 2;
      case '(':return 3;
      case ')':return 4;
      case '$':return 5;
      case 'E':return 6;
      case 'T':return 7;
      case 'F':return 8;
    }
   return -1;
 }
 char decode(int n)
 {
   switch(n)
   {
      case 0:return('i');
      case 1:return('+');
      case 2:return('*');
      case 3:return('(');
      case 4:return(')');
      case 5:return('$');
      case 6:return('E');
      case 7:return('T');
      case 8:return('F');
   }
   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],"E E+T");
  strcpy(pr[2],"E T");
  strcpy(pr[3],"T T*F");
  strcpy(pr[4],"T F");
  strcpy(pr[5],"F (E)");
  strcpy(pr[6],"F i");
  t2[2][1]=t2[2][4]=t2[2][5]=2;
  t2[3][1]=t2[3][2]=t2[3][4]=t2[3][5]=4;
  t2[5][1]=t2[5][2]=t2[5][4]=t2[5][5]=6;
  t2[9][1]=t2[9][4]=t2[9][5]=1;
  t2[10][1]=t2[10][2]=t2[10][4]=t2[10][5]=3;
  t2[11][2]=t2[11][1]=t2[11][4]=t2[11][5]=5;
  t1[2][1]=t1[2][4]=t1[2][5]='r';
  t1[3][1]=t1[3][2]=t1[3][4]='r';
  t1[3][5]=t1[5][1]=t1[5][2]='r';
  t1[5][4]=t1[5][5]=t1[9][1]=t1[9][4]='r';
  t1[9][5]=t1[10][1]=t1[10][2]=t1[10][4]=t1[10][5]='r';
  t1[11][1]=t1[11][4]=t1[11][2]=t1[11][5]='r';
  t1[0][0]=t1[4][0]=t1[6][0]=t1[7][0]=t1[0][3]=t1[4][3]=t1[6][3]='s';
  t1[2][2]=t1[9][2]=t1[8][4]=t1[1][1]=t1[8][1]=t1[7][3]='s';
  t1[1][5]='a';
  t2[0][0]=t2[4][0]=t2[6][0]=t2[7][0]=5;
  t2[0][3]=t2[4][3]=t2[6][3]=t2[7][3]=4;
  t2[2][2]=t2[9][2]=7;
  t2[8][4]=11;
  t2[1][1]=t2[8][1]=6;
  gt[0][6]=1;
  gt[0][7]=gt[4][7]=2;
  gt[0][8]=gt[4][8]=gt[6][8]=3;
  gt[4][6]=8;gt[6][7]=9;gt[7][8]=10;
  printf("enter string:");
  scanf("%s",input);
  for(k=0;input[k];k++)
  {
   inp[k]=encode(input[k]);
   if(input[k]<0||inp[k]>5)
   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");
     }
   getch();
  }
```

INPUT/OUTPUT:

```
enter string:i*(i+i)$

0          i*(i+i)$
0i5          *(i+i)$
0F3          *(i+i)$
0T2          *(i+i)$
0T2*7          (i+i)$
0T2*7(4        i+i)$
0T2*7(4i5       +i)$
0T2*7(4F3       +i)$
0T2*7(4T2       +i)$
0T2*7(4E8       +i)$
0T2*7(4E8+6      i)$
0T2*7(4E8+6i5     )$
0T2*7(4E8+6F3     )$
```

```
0T2*7(4E8+6T9     )$
0T2*7(4E8         )$
0T2*7(4E8)11      $
0T2*7F10          $
0T2               $
0E1               $
string is accepted
```

5. Write a Program to convert the BNF rules into Yacc form & to generate Abstract Syntax Tree

Aim: To convert the BNF rules into Yacc form & to generate Abstract Syntax Tree

Program:

```
<int.l>
%{
#include"y.tab.h"
#include<stdio.h>
#include<string.h>
int LineNo=1;
%}
identifier [a-zA-Z][_a-zA-Z0-9]*
number  [0-9]+|([0-9]*\.[0-9]+)
%%
main\(\) return MAIN;
if return IF;
else return ELSE;
while return WHILE;
int |
char |
float return TYPE;
{identifier}  {strcpy(yylval.var,yytext);
                    return VAR;}

{number}          {strcpy(yylval.var,yytext);
                    return NUM;}

    \< |
    \> |
    \>= |
    \<= |
    ==        {strcpy(yylval.var,yytext);
                    return RELOP;}

    [ \t] ;
    \n  LineNo++;



    return yytext[0];
```

%%

INPUT/OUTPUT:

ADDITIONAL PROGRAM

6. Write a Program to develop a Recursive Descent Parser for the given language

Aim: To develop a Recursive Descent Parser for the given language

Grammer:-

E -> E+T | E-T | T

T -> T*F | T/F | F

F -> (E) | id

To avoid left factoring we have to write the expression grammer as follows

E -> TE'

E' -> +TE' | -TE' | $

T -> FT'

T' -> *FT' | /FT' | $

F -> (E) | id

Program:

```
#include<stdio.h>
char input[20];
int ip=0;
main()
{
  clrscr();
  printf("Enter the string to be parsed :\t");
  scanf("%s",input);
```

```
            E();
            if(input[ip]=='\0')
            printf("Successful Parse");


  else
  printf("Rejected");
  getch();
}
E()
{
 T();
 EPRIME();
}
EPRIME()
{
  if(input[ip] == '+' || input[ip] == '-')
  {
  ip++;
  T();
  EPRIME();
  }
  else
  return;
}
```

```
T()
{
F();
TPRIME();
}
TPRIME()



{
   if(input[ip] == '*' || input[ip] == '/')
   {
     ip++;
     F();
     TPRIME();
   }
   else
   return;
}
F()
{
  if((input[ip] >= 'a' && input[ip] <='z') ||
     (input[ip] >= '0' && input[ip] <= '9'))
  ip++;
  else
  if(input[ip] == '(' )
```

```
  {
  ip++;
  E();
  if(input[ip] == ')' )
  ip++;
  else
  error();
  }
  else


error();
}
error()
{
 printf("\n The String is rejected ");
 getch(); exit(0);
}
```

INPUT/OUTPUT

```
Enter the string to be parsed: 5+3*(a-b)/4
Successful Parse
Enter the string to be parsed : a+((b+c)
The String is rejected
```