OS and CD Lab Manual

Objective:

To provide an understanding of the design aspects of operating system.

To provide an efficient understanding of the language translation peculiarities by designing a complete translator for a mini language.

Recommended Systems/Software Requirements:

- Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space
- Turbo C or TC3 complier in Windows XP or Linux Operating System.

Developed By:

Ms. Shalu Gupta
Asst Prof
CS/IT

Table of Contents

PART A

1) Simulate the following CPU scheduling algorithms

a) FCFS
b) SJF
c) Priority
d) Round Robin

a) FCFS:

AIM: A program to simulate the FCFS CPU scheduling algorithm

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
char pn[10][10];
int arr[10],bur[10],star[10],finish[10],tat[10],wt[10],i,n;
int totwt=0,tottat=0;
clrscr();
printf("Enter the number of processes:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter the Process Name, Arrival Time & Burst Time:");
scanf("%s%d%d",&pn[i],&arr[i],&bur[i]);
}
for(i=0;i<n;i++)
{
if(i==0)
{
star[i]=arr[i];
wt[i]=star[i]-arr[i];
finish[i]=star[i]+bur[i];
tat[i]=finish[i]-arr[i];
}
else
{
star[i]=finish[i-1];
wt[i]=star[i]-arr[i];
finish[i]=star[i]+bur[i];
tat[i]=finish[i]-arr[i];
}
```

```
}
printf("\nPName    Arrtime    Burtime    Start    TAT    Finish");
for(i=0;i<n;i++)
{
printf("\n%s\t%6d\t\t%6d\t%6d\t%6d\t%6d",pn[i],arr[i],bur[i],star[i],tat[i],finish[i]);
totwt+=wt[i];
tottat+=tat[i];
}
printf("\nAverage Waiting time:%f",(float)totwt/n);
printf("\nAverage Turn Around Time:%f",(float)tottat/n);
getch();
}
```

OUTPUT:

Input:

Enter the number of processes: 3
Enter the Process Name, Arrival Time & Burst Time: 1 2 3
Enter the Process Name, Arrival Time & Burst Time: 2 5 6
Enter the Process Name, Arrival Time & Burst Time: 3 6 7

Output:

PName	Arrtime	Burtime	Srart	TAT	Finish
1	2	3	2	3	5
2	5	6	5	6	4
3	6	7	6	7	10

Average Waiting Time: 3.333
Average Turn Around Time: 7.000

b) SJF:

AIM: A program to simulate the SJF CPU scheduling algorithm

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
int et[20],at[10],n,i,j,temp,st[10],ft[10],wt[10],ta[10];
int totwt=0,totta=0;
float awt,ata;
char pn[10][10],t[10];
clrscr();
printf("Enter the number of process:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter process name, arrival time & execution time:");
flushall();
scanf("%s%d%d",pn[i],&at[i],&et[i]);
}
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
if(et[i]<et[j])
{
temp=at[i];
at[i]=at[j];
at[j]=temp;
temp=et[i];
et[i]=et[j];
et[j]=temp;
strcpy(t,pn[i]);
strcpy(pn[i],pn[j]);
strcpy(pn[j],t);
}
}
for(i=0;i<n;i++)
{
if(i==0)
st[i]=at[i];
else
```

```
st[i]=ft[i-1];
wt[i]=st[i]-at[i];
ft[i]=st[i]+et[i];
ta[i]=ft[i]-at[i];
totwt+=wt[i];
totta+=ta[i];
}
awt=(float)totwt/n;
ata=(float)totta/n;
printf("\nPname\tarrivaltime\texecutiontime\twaitingtime\ttatime");
for(i=0;i<n;i++)
printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],wt[i],ta[i]);
printf("\nAverage waiting time is:%f",awt);
printf("\nAverage turnaroundtime is:%f",ata);
getch();
}
```

OUTPUT:

Input:

Enter the number of processes: 3
Enter the Process Name, Arrival Time & Burst Time: 1 4 6
Enter the Process Name, Arrival Time & Burst Time: 2 5 15
Enter the Process Name, Arrival Time & Burst Time: 3 6 11

Output:

Pname	arrivaltime	executiontime	waitingtime	tatime
1	4	6	0	6
3	6	11	4	15
2	5	15	16	31

Average Waiting Time: 6.6667
Average Turn Around Time: 17.3333

c) Priority:

AIM: A program to simulate the priority CPU scheduling algorithm

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
int et[20],at[10],n,i,j,temp,p[10],st[10],ft[10],wt[10],ta[10];
int totwt=0,totta=0;
float awt,ata;
char pn[10][10],t[10];
clrscr();
printf("Enter the number of process:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter process name,arrivaltime,execution time & priority:");
flushall();
scanf("%s%d%d%d",pn[i],&at[i],&et[i],&p[i]);
}
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
if(p[i]<p[j])
{
temp=p[i];
p[i]=p[j];
p[j]=temp;
temp=at[i];
at[i]=at[j];
at[j]=temp;
temp=et[i];
et[i]=et[j];
et[j]=temp;
strcpy(t,pn[i]);
strcpy(pn[i],pn[j]);
strcpy(pn[j],t);
}
}
for(i=0;i<n;i++)
```

```
{
if(i==0)
{
st[i]=at[i];
wt[i]=st[i]-at[i];
ft[i]=st[i]+et[i];
ta[i]=ft[i]-at[i];
}
else
{
st[i]=ft[i-1];
wt[i]=st[i]-at[i];
ft[i]=st[i]+et[i];
ta[i]=ft[i]-at[i];
}
totwt+=wt[i];
totta+=ta[i];
}
awt=(float)totwt/n;
ata=(float)totta/n;
printf("\nPname\tarrivaltime\texecutiontime\tpriority\twaitingtime\ttatime");
for(i=0;i<n;i++)
printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],p[i],wt[i],ta[i]);
printf("\nAverage waiting time is:%f",awt);
printf("\nAverage turnaroundtime is:%f",ata);
getch();
}
```

OUTPUT:

Input:

Enter the number of processes: 3
Enter the Process Name, Arrival Time, execution time & priority: 1 2 3 1
Enter the Process Name, Arrival Time, execution time & priority: 2 4 5 2
Enter the Process Name, Arrival Time, execution time & priority: 3 5 6 3

Output:

Pname	arrivaltime	executiontime	priority	waitingtime	tatime
1	2	3	1	0	3
2	4	5	2	1	6
3	5	6	3	5	11

Average Waiting Time: 2.0000
Average Turn Around Time: 6.6667

d) Round Robin:

AIM: A program to simulate the Round Robin CPU scheduling algorithm

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int et[30],ts,n,i,x=0,tot=0;
char pn[10][10];
clrscr();
printf("Enter the no of processes:");
scanf("%d",&n);
printf("Enter the time quantum:");
scanf("%d",&ts);
for(i=0;i<n;i++)
{
printf("enter process name  & estimated time:");
scanf("%s %d",pn[i],&et[i]);
}
printf("The processes are:");
for(i=0;i<n;i++)
printf("process %d: %s\n",i+1,pn[i]);
for(i=0;i<n;i++)
tot=tot+et[i];
while(x!=tot)
{
for(i=0;i<n;i++)
{
if(et[i]>ts)
{
x=x+ts;
printf("\n %s -> %d",pn[i],ts);

et[i]=et[i]-ts;
}

else
if((et[i]<=ts)&&et[i]!=0)
{
x=x+et[i];
printf("\n %s  ->  %d",pn[i],et[i]);
et[i]=0;}
```

```
}
}
printf("\n Total Estimated Time:%d",x);
getch();
}
```

OUTPUT:

Input:

Enter the no of processes: 2
Enter the time quantum: 3

Enter the process name & estimated time: p1 12
Enter the process name & estimated time: p2 15

Output:

p1 -> 3
p2 -> 3
p1 -> 3
p2 -> 3
p1 -> 3
p2 -> 3
p1 -> 3
p2 -> 3
p2 -> 3

Total Estimated Time: 27

2) Simulate the MVT and MFT.

MVT: multiprocessing with a variable number of tasks
AIM: A program to simulate the MVT.

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("\nenter memory req for process%d: ",i+1);
scanf("%d",&m1);
count=count+m1;
if(m1<=m)
{
if(count==m)
{
printf("there is no further memory remaining:");
}
else
{
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("\nexternal fragmentation for this process is:%d",m2);
}
getch();
}
```

OUTPUT:

MFT : MFT - multiprocessing with a fixed number of tasks

AIM: A Program to simulate the MFT

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int m,p,s,p1;
int m1[4],i,f,f1=0,f2=0,fra1,fra2;
clrscr();
printf("Enter the memory size:");
scanf("%d",&m);
printf("Enter the no of partitions:");
scanf("%d",&p);
s=m/p;
printf("Each partn size is:%d",s);
printf("\nEnter the no of processes:");
scanf("%d",&p1);
for(i=0;i<p1;i++)
{
printf("\nEnter the memory req for process%d:",i+1);
scanf("%d",&m1[i]);
if(m1[i]<=s)
{
printf("\nProcess is allocated in partition%d",i+1);
fra1=s-m1[i];
printf("\nInternal fragmentation for process is:%d",fra1);
f1=f1+fra1;
}
else
{
printf("\nProcess not allocated in partition%d",i+1);
fra2=s;
f2=f2+fra2;
printf("\nExternal fragmentation for partition is:%d",fra2);
}
}
printf("\nProcess\tmemory\tallocatedmemory");
for(i=0;i<p1;i++)
printf("\n%5d\t%5d\t%5d",i+1,s,m1[i]);
f=f1+f2;
printf("\nThe tot no of fragmentation is:%d",f);
getch();
}
```

OUTPUT:

3) Simulate Bankers Algorithm for Deadlock Avoidance.

AIM: A program 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],run[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 each class:");
for(k=1;k<=r;k++)
scanf("%d",&totext[k]);
printf("Enter the allocated resources:");
for(i=1;i<=n;i++)
for(k=1;k<=r;k++)
scanf("%d",&resalloc);
printf("Enter the process making the new request:");
scanf("%d",&p);
printf("Enter the requested resource:");
for(k=1;k<=r;k++)
scanf("%d",&newreq[k]);
printf("Enter the process which are n blocked or running:");
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+resalloc[i][k];
j=totalloc[k];
}
}
for(i=1;i<=n;i++)
{
if(block[i]==1||run[i]==1)
active[i]=1;
else
active[i]=0;
}
for(k=1;k<=r;k++)
{
resalloc[p][k]+=newreq[k];
totalloc[k]+=newreq[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]=totalloc[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]=resalloc[i][k];
}
}
m=0;
for(k=1;k<=r;k++)
resreq[p][k]=newreq[k];
printf("Deadlock willn't occur");
}
else
{
for(k=1;k<=r;k++)
{
resalloc[p][k]=newreq[k];
totalloc[k]=newreq[k];
}
printf("Deadlock will occur");
}
getch();
}
```

OUTPUT:

4) Simulate Bankers Algorithm for Deadlock Prevention.

AIM: A program to simulate Bankers Algorithm for Deadlock Prevention.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int cl[10][10],al[10][10],av[10],i,j,k,m,n,c,ne[10][10],flag=0;
clrscr();
printf("\nEnter the matrix");
scanf("%d %d",&m,&n);
printf("\nEnter the claim matrix");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&cl[i][j]);
}
}
printf("\nEnter allocated matrix");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&al[i][j]);
}
}
printf("\nThe need matrix");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
ne[i][j]=cl[i][j]-al[i][j];
printf("\t%d",ne[i][j]);
}
printf("\n");
}
printf("\nEnter avaliable matrix");
for(i=0;i<3;i++)
scanf("%d",av[i]);
printf("Claim matrix:\n");
for(i=0;i<m;i++)
```

```
{
for(j=0;j<n;j++)
{
printf("\t%d",cl[i][j]);
}
printf("\n");
}
printf("\n allocated matrix:\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
printf("\t%d",al[i][j]);
}
printf("\n");
}
printf(" available matrix:\n");
for(i=0;i<3;i++)
{
printf("\t%d",av[i]);
}
for(k=0;k<m;k++)
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
if(av[j]>=ne[i][j])
flag=1;
else
break;
if(flag==1&& j==n-1)
goto a;
}
}
a: if(flag==0)
{
printf("unsafestate");
}
if(flag==1)
{
flag=0;
for(i=0;i<m;i++)
{
for(j=0;j<n;i++)
{
av[j]+=al[i][j];
```

```
al[i][j]=1;
}
}
printf("\n safe state");
for(i=0;i<n;i++)
printf("\t available matrix:%d",av[i]);
}
getch();
}
```

OUTPUT:

5) Simulate all Page Replacement Algorithms

a) **FIFO**
b) **LRU**
c) **Optimal**

a) FIFO:

AIM: A program to simulate FIFO Page Replacement Algorithm

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[5],b[20],p=0,q=0,m=0,h,k,i,q1=1,j,u;
char f='F';
clrscr();
printf("Enter numbers:");
for(i=0;i<12;i++)
scanf("%d",&b[i]);
for(i=0;i<12;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)&&(q1==3))
{
printf("-->%c",f);
m++;
}
p=0;
for(k=0;k<q-1;k++)
{
```

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

OUTPUT:

b) LRU:

AIM: A program 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;
char f='F';
clrscr();
printf("Enter no:");
for(i=0;i<12;i++)
scanf("%d",&b[i]);
for(i=0;i<12;i++)
{if(p==0)
{
if(q>=3)
q=0;
a[q]=b[i];
q++;
if(q1<3)
{
q1=q;
g=1;
}
}
printf("\n%d",b[i]);
printf("\t");
for(h=0;h<q1;h++)
printf("%d",a[h]);
if((p==0)&&(q1==3)&&(g!=1))
{
printf("-->%c",f);
m++;
}
p=0;
g=0;
if(q1==3)
{
for(k=0;k<q-1;k++)
{
if(b[i+1]==a[k])
p=1;
```

```
}
for(j=0;j<q1;j++)
{
u=0;
k=i;
while(k>(i-2)&&(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();

}
```

OUTPUT:

c) Optimal:

AIM: A program to simulate Optimal Page Replacement Algorithm.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int pn[12],m[3]={0,0,0},m1[3],i,j,k;
int flag,f,pf=0,z;
clrscr();
printf("enter pgs:");
for(i=0;i<12;i++)
scanf("%d",&pn[i]);
j=0;
for(i=0;i<3;i++)
{
while(j<12)
{
flag=0;for(k=0;k<3;k++)
{
if(m[k]==pn[j])
{
flag=1;
j++;
i--;
}
if(flag==1)
break;
}
if(flag==0)
{
m[j]=pn[j];
flag=1;
}
j++;
if(flag==1)
break;
}
}
for(i=j;i<12;i++)
{
flag=0;
```

```
for(j=0;j<3;j++)
{
if(pn[i]==m[j])
flag=1;
if(flag==0)
{
m1[0]=0;
m1[1]=m1[2]=0;
for(j=0;j<3;j++)
{f=0;
for(k=i+1;k<12;k++)
{
if(m[j]==pn[k])
{
m1[j]=k;
f=1;
}
if(f==1)
break;
}
}
z=(m1[0]>m1[1]||(m1[0]>m1[2])?m1[0]:m1[2])&&(m1[1]>m1[2]?m1[1]:m1[2]);
for(j=0;j<3;j++)
{
if(pn[z]==m[j])
{
m[j]=pn[i];
pf++;
}
}
}
}
}
printf("no of faults:%d",pf);
getch();
}
```

OUTPUT:

6) Simulate Paging technique of Memory Management.

AIM: A program to simulate Paging technique of memory management.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
main()
{
 int np,ps,i;
 int *sa;
 clrscr();
 printf("enter how many pages\n");
 scanf("%d",&np);
 printf("enter the page size \n");
 scanf("%d",&ps);
 sa=(int*)malloc(2*np);
 for(i=0;i<np;i++)
 {
  sa[i]=(int)malloc(ps);
  printf("page%d\t address %u\n",i+1,sa[i]);
 }
getch();
}
```

OUTPUT:

PART B

7) Write a program to design lexical analyzer.

AIM: A program to design Lexical Analyzer.

PROGRAM:

```
#include<string.h>
#include<ctype.h>
#include<stdio.h>
void keyword(char str[10])
{
if(strcmp("for",str)==0||strcmp("while",str)==0||strcmp("do",str)==0||
strcmp("int",str)==0||strcmp("float",str)==0||strcmp("char",str)==0||
strcmp("double",str)==0||strcmp("static",str)==0||strcmp("switch",str)==0||
strcmp("case",str)==0)
printf("\n%s is a keyword",str);
else
 printf("\n%s is an identifier",str);
}
main()
{
 FILE *f1,*f2,*f3;
char c,str[10],st1[10];
int num[100],lineno=0,tokenvalue=0,i=0,j=0,k=0;
      printf("\nEnter the c program");/*gets(st1);*/
f1=fopen("input","w");
while((c=getchar())!=EOF)
putc(c,f1);
fclose(f1);
f1=fopen("input","r");
f2=fopen("identifier","w");
f3=fopen("specialchar","w");
while((c=getc(f1))!=EOF)
{
 if(isdigit(c))
{
 tokenvalue=c-'0';
c=getc(f1);
while(isdigit(c))
{
 tokenvalue*=10+c-'0';
c=getc(f1);
}
num[i++]=tokenvalue;
ungetc(c,f1);
}
else if(isalpha(c))
```

```
{
 putc(c,f2);
c=getc(f1);
while(isdigit(c)||isalpha(c)||c=='_'||c=='$')
{
 putc(c,f2);
c=getc(f1);
}
putc(' ',f2);
ungetc(c,f1);
}
else if(c==' '||c=='\t')
printf(" ");

else if(c=='\n')
lineno++;
else
 putc(c,f3);
}
fclose(f2);
fclose(f3);
fclose(f1);
printf("\nThe no's in the program are");
for(j=0;j<i;j++)
printf("%d",num[j]);
printf("\n");
f2=fopen("identifier","r");
k=0;
printf("The keywords and identifiersare:");
while((c=getc(f2))!=EOF)
{
 if(c!=' ')
str[k++]=c;
else
{
     str[k]='\0';
     keyword(str);
     k=0;
}
}
fclose(f2);
f3=fopen("specialchar","r");
printf("\nSpecial characters are");
while((c=getc(f3))!=EOF)
printf("%c",c);
printf("\n");
```

```
fclose(f3);
printf("Total no. of lines are:%d",lineno);
}
```

OUTPUT:

```
Enter the C program
a+b*c

Ctrl-D

The no's in the program are:
The keywords and identifiers are:
a is an identifier and terminal
b is an identifier and terminal
c is an identifier and terminal
Special characters are:
+ *

Total no. of lines are: 1*/
```

8) Write a program to implement the lexical analyzer using lex tool.

AIM: A program to implement the Lexical Analyzer.

PROGRAM:

```
/* program name is lexp.l */
%{
  /* program to recognize a c program */
    int COMMENT=0;
%}
identifier [a-zA-Z][a-zA-Z0-9]*
%%

#.* { printf("\n%s is a PREPROCESSOR DIRECTIVE",yytext);}
int |
float |
char |
double |
while |
for |
do |
if  |
break |
continue |
void |
switch |
case |
long |
struct |
const |
typedef |
return |
else |
goto      {printf("\n\t%s is a KEYWORD",yytext);}
"/*" {COMMENT = 1;}
    /*{printf("\n\n\t%s is a COMMENT\n",yytext) ;}*/
"*/" {COMMENT = 0;}
    /* printf("\n\n\t%s is a COMMENT\n",yytext);}*/
{identifier}\(   {if(!COMMENT)printf("\n\nFUNCTION\n\t%s",yytext);}
\{  {if(!COMMENT) printf("\n BLOCK BEGINS");}

\}  {if(!COMMENT) printf("\n BLOCK ENDS");}

{identifier}(\[[0-9]*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}
```

```
\".*\"  {if(!COMMENT) printf("\n\t%s is a STRING",yytext);}

[0-9]+  {if(!COMMENT) printf("\n\t%s is a NUMBER",yytext);}
\)(\;)? {if(!COMMENT) printf("\n\t");ECHO;printf("\n");}

\(          ECHO;
=       {if(!COMMENT)printf("\n\t%s is an ASSIGNMENT OPERATOR",yytext);}

\<= |
\>= |
\< |
== |
\>     {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}

%%
int main(int argc,char **argv)
{
 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:

```
/*Input:

$vi var.c

#include<stdio.h>
main()
{
  int a,b;
```

```
}
```

Output:

```
$lex lex.l
$cc lex.yy.c
$./a.out var.c
#include<stdio.h> is a PREPROCESSOR DIRECTIVE

FUNCTION

   main (

       )

BLOCK BEGINS
   int is a KEYWORD

a IDENTIFIER

b IDENTIFIER
BLOCK ENDS*/
```

9) Design predictive parser for the given language.

AIM: A program to implementation of Predictive Parser.

PROGRAM:

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
#include<stdlib.h>
#define SIZE 128
#define NONE -1
#define EOS '\0'
#define NUM 257
#define KEYWORD 258
#define ID 259
#define DONE 260
#define MAX 999
char lexemes[MAX];
char buffer[SIZE];
int lastchar=-1;
int lastentry=0;
int tokenval=DONE;
int lineno=1;
int lookahead;
struct entry
{
  char *lexptr;
  int token;
}symtable[100];
struct entry
keywords[]={"if",KEYWORD,"else",KEYWORD,"for",KEYWORD,"int",KEYWORD,
"float",KEYWORD,"double",KEYWORD,"char",KEYWORD,"struct",KEYWORD,"ret
urn",KEYWORD,0,0};
void Error_Message(char *m)
{
  fprintf(stderr,"line %d, %s \n",lineno,m);
  exit(1);
}
int look_up(char s[ ])
{
  int k;
  for(k=lastentry;k>0;k--)
    if(strcmp(symtable[k].lexptr,s)==0)
      return k;
```

```
    return 0;
}
int insert(char s[ ],int tok)
{
  int len;
  len=strlen(s);
  if(lastentry+1>=MAX)
    Error_Message("Symbpl table is full");
  if(lastchar+len+1>=MAX)
    Error_Message("Lexemes array is full");
  lastentry=lastentry+1;
  symtable[lastentry].token=tok;
  symtable[lastentry].lexptr=&lexemes[lastchar+1];
  lastchar=lastchar+len+1;
  strcpy(symtable[lastentry].lexptr,s);
  return lastentry;
}
/*void Initialize()
{
  struct entry *ptr;
  for(ptr=keywords;ptr- >token;ptr+1)
    insert(ptr->lexptr,ptr->token);
}*/
int lexer()
{
  int t;
  int val,i=0;
  while(1)
  {
    t=getchar();
    if(t==' '||t=='\t');
    else if(t=='\n')
    lineno=lineno+1;
    else if(isdigit(t))
    {
      ungetc(t,stdin);
      scanf("%d",&tokenval);
      return NUM;
    }
    else if(isalpha(t))
    {
      while(isalnum(t))
      {
        buffer[i]=t;
        t=getchar();
        i=i+1;
```

```
        if(i>=SIZE)
          Error_Message("Compiler error");
       }
       buffer[i]=EOS;
       if(t!=EOF)
       ungetc(t,stdin);
       val=look_up(buffer);
       if(val==0)
         val=insert(buffer,ID);
       tokenval=val;
       return symtable[val].token;
     }
     else if(t==EOF)
       return DONE;
     else
     {
       tokenval=NONE;
       return t;
     }
   }
}
void Match(int t)
{
  if(lookahead==t)
    lookahead=lexer();
  else
    Error_Message("Syntax error");
}
void display(int t,int tval)
{
  if(t=='+'||t=='-'||t=='*'||t=='/')
    printf("\nArithmetic Operator: %c",t);
  else if(t==NUM)
    printf("\n Number: %d",tval);
  else if(t==ID)
    printf("\n Identifier : %s",symtable[tval].lexptr);
  else
    printf("\n Token %d tokenval %d",t,tokenval);
}
void F()
{
  //void E();
  switch(lookahead)
  {
    case '(' : Match('(');
               E();
```

```
            Match(')');
            break;
   case NUM : display(NUM,tokenval);
            Match(NUM);
            break;
   case ID :  display(ID,tokenval);
            Match(ID);
            break;
   default :  Error_Message("Syntax error");
  }
}
void T()
{
  int t;
  F();
  while(1)
  {
    switch(lookahead)
    {
      case '*' : t=lookahead;
                 Match(lookahead);
                 F();
                 display(t,NONE);
                 continue;
      case '/' : t=lookahead;
                 Match(lookahead);
                 display(t,NONE);
                 continue;
      default :  return;
    }
  }
}
void E()
{
  int t;
  T();
  while(1)
  {
    switch(lookahead)
    {
      case '+' : t=lookahead;
                 Match(lookahead);
                 T();
                 display(t,NONE);
                 continue;
      case '-' : t=lookahead;
```

```
            Match(lookahead);
            T();
            display(t,NONE);
            continue;
     default :  return;
    }
  }
}
void parser()
{
  lookahead=lexer();
  while(lookahead!=DONE)
  {
    E();
    Match(';');
  }
}
main()
{
  char ans[10];
  printf("\n Program for recursive decent parsing ");
  printf("\n Enter the expression ");
  printf("And place ; at the end\n");
  printf("Press Ctrl-Z to terminate\n");
  parser();
}
```

OUTPUT:

```
Program for recursive decent parsing
Enter the expression And place ; at the end
Press Ctrl-Z to terminate
a+b*c;
Identifier: a
Identifier: b
Identifier: c
Arithmetic Operator: *
Arithmetic Operator: +
2*3;
Number: 2
Number: 3
Arithmetic Operator: *
+3;
line 5,Syntax error
Ctrl-Z
```

10) Design LALR Bottom up Parser

AIM: A program to design LALR Bottom up Parser.

PROGRAM:

<parser.l>

```
%{
 #include<stdio.h>
#include "y.tab.h"
%}
%%
[0-9]+ {yylval.dval=atof(yytext);
return DIGIT;
}
\n|. return yytext[0];
%%
```

<parser.y>

```
%{
 /*This YACC specification file generates the LALR parser for the program
considered in experiment 4.*/
       #include<stdio.h>
%}
%union
{
 double dval;
}

%token <dval> DIGIT
%type <dval> expr
%type <dval> term
%type <dval> factor

%%

line: expr '\n'        {
 printf("%g\n",$1);
}
;
expr: expr '+' term {$$=$1 + $3 ;}
    | term
    ;
term: term '*' factor  {$$=$1 * $3 ;}
    | factor
```

```
    ;
factor: '(' expr ')' {$$=$2 ;}

    | DIGIT
    ;

%%
int main()
{
 yyparse();
}
yyerror(char *s)
{
 printf("%s",s);
}
```

Output:

```
$lex parser.l
$yacc –d parser.y
$cc lex.yy.c y.tab.c –ll –lm
$./a.out
2+3
5.0000*/
```

11) Convert the BNF rules into YACC form and write code to generate abstract syntax tree.

AIM: A program to Convert the BNF rules into YACC form and write code 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];
%%
                  < int.y>
```

```
%{
#include<string.h>
#include<stdio.h>
struct quad
{
  char op[5];
  char arg1[10];
  char arg2[10];
  char result[10];
}QUAD[30];
struct stack
{
  int items[100];
  int top;
}stk;
int Index=0,tIndex=0,StNo,Ind,tInd;
extern int LineNo;
%}
%union
{
  char var[10];
}
%token <var> NUM VAR RELOP
%token MAIN IF ELSE WHILE TYPE

%type <var> EXPR ASSIGNMENT CONDITION IFST ELSEST WHILELOOP
%left '-' '+'
%left '*' '/'
%%


PROGRAM :  MAIN BLOCK
;
BLOCK: '{' CODE '}'
;

CODE: BLOCK
     | STATEMENT CODE
     | STATEMENT
;
STATEMENT: DESCT ';'
          | ASSIGNMENT ';'
          | CONDST
          | WHILEST
;
DESCT: TYPE VARLIST
```

```
;

VARLIST: VAR ',' VARLIST
       | VAR
;

ASSIGNMENT: VAR '=' EXPR{
                    strcpy(QUAD[Index].op,"=");
                    strcpy(QUAD[Index].arg1,$3);
                    strcpy(QUAD[Index].arg2,"");
                    strcpy(QUAD[Index].result,$1);
                    strcpy($$,QUAD[Index++].result);
                   }
;
EXPR:  EXPR '+' EXPR {AddQuadruple("+",$1,$3,$$);}
      | EXPR '-' EXPR {AddQuadruple("-",$1,$3,$$);}
      | EXPR '*' EXPR {AddQuadruple("*",$1,$3,$$);}
      | EXPR '/' EXPR {AddQuadruple("/",$1,$3,$$);}
      | '-' EXPR {AddQuadruple("UMIN",$2,"",$$);}
      | '(' EXPR ')' {strcpy($$,$2);}
      | VAR
      | NUM
;
CONDST: IFST{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
}
|  IFST ELSEST
;
IFST: IF '(' CONDITION ')' {
strcpy(QUAD[Index].op,"==");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"FALSE");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}

BLOCK {
strcpy(QUAD[Index].op,"GOTO");
strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
push(Index);
```

```
Index++;
}
;

ELSEST: ELSE{
tInd=pop();
Ind=pop();
push(tInd);
sprintf(QUAD[Ind].result,"%d",Index);
}
BLOCK{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
}
;
CONDITION: VAR R ELOP VAR {AddQuadruple($2,$1,$3,$$);
                    StNo=Index-1;
                  }
                  | VAR
                  | NUM
                  ;
WHILEST: WHILELOOP{
                Ind=pop();
                sprintf(QUAD[Ind].result,"%d",StNo);
                Ind=pop();
                sprintf(QUAD[Ind].result,"%d",Index);
               }
;
WHILELOOP: WHILE '(' CONDITION ')' {
                      strcpy(QUAD[Index].op,"==");
                      strcpy(QUAD[Index].arg1,$3);
                      strcpy(QUAD[Index].arg2,"FALSE");
                      strcpy(QUAD[Index].result,"-1");
                      push(Index);
                      Index++;
                     }
BLOCK {
    strcpy(QUAD[Index].op,"GOTO");
    strcpy(QUAD[Index].arg1,"");
    strcpy(QUAD[Index].arg2,"");
    strcpy(QUAD[Index].result,"-1");
    push(Index);
    Index++;
    }
 ;
%%
```

```
extern FILE *yyin;
int main(int argc,char *argv[])
{
  FILE *fp;
  int i;
  if(argc>1)
  {
    fp=fopen(argv[1],"r");
    if(!fp)
    {
      printf("\n File not found");
      exit(0);
    }
    yyin=fp;
  }
  yyparse();
  printf("\n\n\t\t ----------------------------""\n\t\t Pos Operator Arg1 Arg2 Result" "\n\t\t
--------------------");
  for(i=0;i<Index;i++)
  {
    printf("\n\t\t %d\t %s\t %s\t %s\t
%s",i,QUAD[i].op,QUAD[i].arg1,QUAD[i].arg2,QUAD[i].result);
  }
  printf("\n\t\t ------------------------");
  printf("\n\n");
  return 0;
}
void push(int data)
{
  stk.top++;
  if(stk.top==100)
  {
    printf("\n Stack overflow\n");
    exit(0);
  }
  stk.items[stk.top]=data;
}
int pop()
{
  int data;
  if(stk.top==-1)
  {
    printf("\n Stack underflow\n");
    exit(0);
  }
  data=stk.items[stk.top--];
```

```
  return data;
}
void AddQuadruple(char op[5],char arg1[10],char arg2[10],char result[10])
{
  strcpy(QUAD[Index].op,op);
  strcpy(QUAD[Index].arg1,arg1);
  strcpy(QUAD[Index].arg2,arg2);
  sprintf(QUAD[Index].result,"t%d",tIndex++);
  strcpy(result,QUAD[Index++].result);
}
yyerror()
{
  printf("\n Error on line no:%d",LineNo);
}
```

Output:

Input:

```
$vi test.c
main()
{
  int a,b,c;
  if(a<b)
  {
    a=a+b;
  }
  while(a<b)
  {
    a=a+b;
  }
  if(a<=b)
  {
    c=a-b;
  }
  else
  {
    c=a+b;
  }
}
```

Output:

```
$lex int.l
$yacc –d int.y
$gcc lex.yy.c y.tab.c –ll –lm
$./a.out test.c
```

Pos Operator Arg1 Arg2 Result

0 < a b to

1 == to FALSE 5

2 + a b t1
3 = t1 a

4 GOTO 5
5 < a b t2

6 == t2 FALSE 10

7 + a b t3
8 = t3 a

9 GOTO 5
10 <= a b t4

11 == t4 FALSE 15

12 - a b t5
13 = t5 c

14 GOTO 17
15 + a b t3
16 = t6 c
*/

12) A Program to Generate Machine Code.

AIM: A Program to Generate Machine Code.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int label[20];
int no=0;
int main()
{
  FILE *fp1,*fp2;
  char fname[10],op[10],ch;
  char operand1[8],operand2[8],result[8];
  int i=0,j=0;
  printf("\n Enter filename of the intermediate code");
  scanf("%s",&fname);
  fp1=fopen(fname,"r");
  fp2=fopen("target.txt","w");
  if(fp1==NULL || fp2==NULL)
  {
    printf("\n Error opening the file");
   exit(0);
  }
  while(!feof(fp1))
  {
    fprintf(fp2,"\n");
    fscanf(fp1,"%s",op);
    i++;
    if(check_label(i))
      fprintf(fp2,"\nlabel#%d",i);
    if(strcmp(op,"print")==0)
    {
      fscanf(fp1,"%s",result);
      fprintf(fp2,"\n\t OUT %s",result);
    }
    if(strcmp(op,"goto")==0)
    {
      fscanf(fp1,"%s %s",operand1,operand2);
      fprintf(fp2,"\n\t JMP %s,label#%s",operand1,operand2);
      label[no++]=atoi(operand2);
    }
    if(strcmp(op,"[]=")==0)
    {
```

```
 fscanf(fp1,"%s %s %s",operand1,operand2,result);
 fprintf(fp2,"\n\t STORE %s[%s],%s",operand1,operand2,result);
}
if(strcmp(op,"uminus")==0)

{
 fscanf(fp1,"%s %s",operand1,result);
 fprintf(fp2,"\n\t LOAD -%s,R1",operand1);
 fprintf(fp2,"\n\t STORE R1,%s",result);
}
switch(op[0])
{
 case '*':  fscanf(fp1,"%s %s %s",operand1,operand2,result);
            fprintf(fp2,"\n \t LOAD",operand1);
            fprintf(fp2,"\n \t LOAD %s,R1",operand2);
            fprintf(fp2,"\n \t MUL R1,R0");
            fprintf(fp2,"\n \t STORE R0,%s",result);
            break;
 case '+':  fscanf(fp1,"%s %s %s",operand1,operand2,result);
            fprintf(fp2,"\n \t LOAD %s,R0",operand1);
            fprintf(fp2,"\n \t LOAD %s,R1",operand2);
            fprintf(fp2,"\n \t ADD R1,R0");
            fprintf(fp2,"\n \t STORE R0,%s",result);
            break;
 case '-':  fscanf(fp1,"%s %s %s",operand1,operand2,result);
            fprintf(fp2,"\n \t LOAD %s,R0",operand1);
            fprintf(fp2,"\n \t LOAD %s,R1",operand2);
            fprintf(fp2,"\n \t SUB R1,R0");
            fprintf(fp2,"\n \t STORE R0,%s",result);
            break;
 case '/':  fscanf(fp1,"%s %s %s",operand1,operand2,result);
            fprintf(fp2,"\n \t LOAD %s,R0",operand1);
            fprintf(fp2,"\n \t LOAD %s,R1",operand2);
            fprintf(fp2,"\n \t DIV R1,R0");
            fprintf(fp2,"\n \t STORE R0,%s",result);
            break;
 case '%':  fscanf(fp1,"%s %s %s",operand1,operand2,result);
            fprintf(fp2,"\n \t LOAD %s,R0",operand1);
            fprintf(fp2,"\n \t LOAD %s,R1",operand2);
            fprintf(fp2,"\n \t DIV R1,R0");
            fprintf(fp2,"\n \t STORE R0,%s",result);
            break;
 case '=':  fscanf(fp1,"%s %s",operand1,result);
            fprintf(fp2,"\n\t STORE %s %s",operand1,result);
            break;
 case '>':  j++;
```

```
            fscanf(fp1,"%s %s %s",operand1,operand2,result);
            fprintf(fp2,"\n \t LOAD %s,R0",operand1);
            fprintf(fp2,"\n\t JGT %s,label#%s",operand2,result);
            label[no++]=atoi(result);
            break;
    case '<':  fscanf(fp1,"%s %s %s",operand1,operand2,result);
            fprintf(fp2,"\n \t LOAD %s,R0",operand1);
            fprintf(fp2,"\n\t JLT %s,label#%d",operand2,result);
            label[no++]=atoi(result);
            break;
   }
  }
  fclose(fp2);
  fclose(fp1);
  fp2=fopen("target.txt","r");
  if(fp2==NULL)
  {
   printf("Error opening the file\n");
   exit(0);
  }
  do
  {
   ch=fgetc(fp2);
   printf("%c",ch);
  }while(ch!=EOF);
  fclose(fp1);
  return 0;
}
int check_label(int k)
{
  int i;
  for(i=0;i<no;i++)
  {
   if(k==label[i])
     return 1;
  }
  return 0;
}
```

Output:

Input:

```
$vi int.txt
=t1 2
[]=a 0 1
[]=a 1 2
```

```
[]=a 2 3
*t1 6 t2
+a[2] t2 t3
-a[2] t1 t2
/t3 t2 t2
uminus t2 t2
print t2
goto t2 t3
=t3 99
uminus 25 t2
*t2 t3 t3
uminus t1 t1
+t1 t3 t4
print t4
```

Output:

```
Enter filename of the intermediate code: int.txt

STORE t1,2
STORE a[0],1
STORE a[1],2
STORE a[2],3

LOAD t1,R0
LOAD 6,R1
ADD R1,R0
STORE R0,t3

LOAD a[2],R0
LOAD t2,R1
ADD R1,R0
STORE R0,t3

LOAD a[t2],R0
LOAD t1,R1
SUB  R1,R0
STORE R0,t2

LOAD t3,R0
LOAD t2,R1
DIV R 1,R0
STORE R0,t2
LOAD t2,R1
STORE R1,t2
LOAD t2,R0
JGT 5,label#11
```

```
Label#11: OUT t2
        JMP t2,label#13
Label#13: STOR E t3,99
        LOAD 25,R1
        STORE R1,t2

        LOAD t2,R0
        LOAD t3,R1
        MUL R1,R0
        STORE R0,t3

        LOAD t1,R1
        STORE R1,t1

        LOAD t1,R0
        LOAD t3,R1
        ADD R1,R0
        STORE R0,t4
        OUT t4
```