St. Thomas College of Engineering & Technology

Sivapuram(P.O.), Mattanur, Kannur- 670702
(Approved by Govt. of Kerala, Affiliated to APJ Abdul Kalam University)

<u>Department of Computer science and Engineering</u>

LABORATORY MANUAL

CSL331 System Software Lab

PREPARED BY

SREEREKHA K P

Asst. Professor, Department of Computer Science & Engineering

SYLLABUS

SYSTEM SOFTWARE LAB: List of Exercises/ Experiments

(Minimum 8 Exercises (at least 3 and 5 questions from each part V and VI)) : 2 Hrs/week

A. Exercises/Experiments from operating system

1. Simulate the following non-preemptive CPU scheduling algorithms to find turnaround

time and waiting time.

a) FCFS b) SJF c) Round Robin (pre-emptive) d) Priority

2. Simulate the following file allocation strategies.

a) Sequential b) Indexed c) Linked

3. Implement the different paging techniques of memory management.

4. Simulate the following file organization techniques

a) Single level directory b) Two level directory c) Hierarchical

5. Implement the banker's algorithm for deadlock avoidance.

6. Simulate the following disk scheduling algorithms.

a) FCFS b) SCAN c) C-SCAN

7. Simulate the following page replacement algorithms:

a)FIFO b)LRU c) LFU

B. Exercises/Experiments from assemblers, loaders and macroprocessor

1. Implement pass one of a two pass assembler.

2. Implement pass two of a two pass assembler.

3. Implement a single pass assembler.

4. Implement a two pass macro processor

5.Implement a single pass macro processor.

6. Implement an absolute loader.

7. Implement a relocating loader

CONTENTS

EXPERIMENT NO : 1

CPU SCHEDULING ALGORITHMS

AIM

To simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time.

a) FCFS b) SJF c) Round Robin (pre-emptive) d) Priority

PROGRAM

a) FCFS – First Come First Served

```
        #include<stdio.h>

        void main()

{

        int i=0,j=0,b[i],g[20],p[20],w[20],t[20],a[20],n=0,m;

        float avgw=0,avgt=0;

        printf("\n FCFS Scheduling...\n");

        printf("Enter the number of process : ");

        scanf("%d",&n);

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

        {

                printf("Process ID : ");

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

```
    printf("Burst Time : ");
    scanf("%d",&b[i]);


    printf("Arrival Time: ");
    scanf("%d",&a[i]);
}


int temp=0;
for(i=0;i<n-1;i++)
{
    for(j=0;j<n-1;j++)
    {
            if(a[j]>a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;


                temp=b[j];
                b[j]=b[j+1];
                b[j+1]=temp;
                temp=p[j];
                p[j]=p[j+1];
                p[j+1]=temp;
            }
    }
}

g[0]=0;
for(i=0;i<=n;i++)
    g[i+1]=g[i]+b[i];
for(i=0;i<n;i++)
{

    t[i]=g[i+1]-a[i];
    w[i]=t[i]-b[i];
    avgw+=w[i];
    avgt+=t[i];
}
avgw=avgw/n;
avgt=avgt/n;
printf("\n\n Process Scheduling....\n”);
```

```
	printf("Process\tArrivalTime\tBrustTime\tCompletionTime\tWaitingTime\tTurnaroundTime\n");

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

	{

		printf("%d\t%d\t%d\t%d\t\t%d\t\t\t%d\n",p[i],a[i],b[i],g[i+1],w[i],t[i]);

	}

	printf("\nAverage Waiting Time %f",avgw);

	printf("\nAverage Turnaround Time %f",avgt);

}
```

OUTPUT

```
FCFS Scheduling…..

Enter  the number of processes : 3

Process ID : 1

Burst Time : 24

Arrival Time: 0

Process ID : 2

Burst Time : 3

Arrival Time: 0

Process ID : 3

Burst Time : 3
```

```
Arrival Time: 0

Process Scheduling……


Process   ArrivalTime  BurstTime  CompletionTime  WaitingTime  TurnaroundTime

1              0            24           24              0                24

2              0            3            27              24               27

3              0            3            30              27               30


Average Waiting Time : 17


Average Turnaround Time : 27
```

PROGRAM

b) SJF – Shortest Job First

```
#include<stdio.h>

#include<stdlib.h>

typedef struct

{

 int pid; int btime; int wtime;


} sp;
```

```
int main()
{
 int i,j,n,tbm=0,totwtime=0,totttime=0;
 sp *p,t;
 printf("\n SJF Scheduling ..\n");
 printf("Enter the no of process: ");
 scanf("%d",&n);
 p=(sp*)malloc(sizeof(sp));
 printf("\n enter the burst time");
 for(i=0;i<n;i++)
 {
 printf("\n process %d\t",i+1);
 scanf("%d",&p[i].btime);
 p[i].pid=i+1;
 p[i].wtime=0;
 }
 for(i=0;i<n;i++)
 {
 for(j=i+1;j<n;j++)
 {
 if(p[i].btime > p[j].btime)
 {
 t=p[i];
 p[i]=p[j];
 p[j]=t;
 }
 }
 }

 printf("\n process scheduling\n");
 printf("\n Process \tBurst time \t Waiting time ");
 for(i=0;i<n;i++)
 {
 totwtime+=p[i].wtime=tbm;
 tbm+=p[i].btime;
 printf("\n\t%d\t\t\t%d",p[i].pid,p[i].btime);
 printf("\t\t\t%d",p[i].wtime);

 }
 totttime=tbm+totwtime;
 printf("\n Total waiting time :%d", totwtime );
 printf("\n Average waiting time :%f",(float)totwtime/n);
 printf("\n Total turn around time :%d",totttime);
 printf("\n Average turn around time: :%f",(float)totttime/n);
}
```

```
SJF scheduling…..
Enter the number of processes : 4
Enter the burst time :
```

```
Process 1 6

Process 2 8

Process 3 7

Process 4 3

Process Scheduling……

Process Burst time Waiting Time

4          3          0

1          6          3

3          7          9

2           8         16

Total waiting time : 28

Average waiting time : 7

Total turn around time : 52

Average turn around time :13
```

c) Round Robin (pre-emptive)

```
#include<stdio.h>

 int main()

{

    int i, limit, total = 0, x, counter = 0, time_quantum;

    int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10],
temp[10];

    float average_wait_time, average_turnaround_time;

    printf("\nEnter Total Number of Processes:\t");
```

```
scanf("%d", &limit);
x = limit;
for(i = 0; i < limit; i++)
{
    printf("\nEnter Details of Process[%d]\n", i + 1);
    printf("Arrival Time:\t");
    scanf("%d", &arrival_time[i]);
    printf("Burst Time:\t");
    scanf("%d", &burst_time[i]);
    temp[i] = burst_time[i];
}
printf("\nEnter Time Quantum:\t");
scanf("%d", &time_quantum);
printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");
for(total = 0, i = 0; x != 0;)
{
    if(temp[i] <= time_quantum && temp[i] > 0)
    {
        total = total + temp[i];
        temp[i] = 0;
        counter = 1;
    }
    else if(temp[i] > 0)
    {
            temp[i] = temp[i] - time_quantum;

            total = total + time_quantum;

        }

        if(temp[i] == 0 && counter == 1)

        {

            x--;

            printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d", i + 1, burst_time[i], total - arrival_time[i], total - arrival_time[i] - burst_time[i]);

            wait_time = wait_time + total - arrival_time[i] - burst_time[i];

            turnaround_time = turnaround_time + total - arrival_time[i];

            counter = 0;

        }

        if(i == limit - 1)

        {

            i = 0;

        }

        else if(arrival_time[i + 1] <= total)

        {

            i++;

        }

        else

        {

            i = 0;

        }
```

```
    }
    average_wait_time = wait_time * 1.0 / limit;
    average_turnaround_time = turnaround_time * 1.0 / limit;
    printf("\n\nAverage Waiting Time:\t%f", average_wait_time);
    printf("\nAvg Turnaround Time:\t%f\n", average_turnaround_time);
    return 0;
}
```

OUTPUT

```
Enter Total Number of Processes: 4


Enter Details of Process[1]
Arrival Time:0
Burst Time:  9


Enter Details of Process[2]
Arrival Time:1
Burst Time:  5


Enter Details of Process[3]
Arrival Time:        2
Burst Time:  3
```

```
Enter Details of Process[4]

Arrival Time:       3

Burst Time:  4


Enter Time Quantum:     5


Process ID          Burst Time    Turnaround Time   Waiting Time


Process[2]          5             9                 4

Process[3]          3             11                8

Process[4]          4             14                10

Process[1]          9             21                12


Average Waiting Time:   8.500000

Avg Turnaround Time:    13.750000
```

d) Priority

```
#include<stdio.h>

#include<stdio.h>

#include<stdlib.h>

typedef struct

{

 int pno;
int pri;

int btime;

int wtime;

}sp;


int main()

{

int i,j,n;

int tbm=0,totwtime=0,totttime=0;

sp *p,t;

printf("\n PRIORITY SCHEDULING.\n");

printf("\n Enter the no of process : \n");

scanf("%d",&n);

p=(sp*)malloc(sizeof(sp));

printf(" Enter the burst time and priority\n");

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

{

printf(" process %d : \n", i+1);

scanf("%d%d",&p[i].btime,&p[i].pri);

p[i].pno=i+1;

p[i].wtime=0;

}

for(i=0;i<n-1;i++)

for(j=i+1;j<n;j++)
```

```
{
if(p[i].pri>p[j].pri)
{
t=p[i];
p[i]=p[j];
p[j]=t;
}
}
printf("\n process\tbursttime\twaiting time\tturnaround time\n");
for(i=0;i<n;i++)
{
totwtime+=p[i].wtime=tbm;
tbm+=p[i].btime;
printf("\n\t%d\t\t\t%d",p[i].pno,p[i].btime);
printf("\t\t\t%d\t\t\t%d",p[i].wtime,p[i].wtime+p[i].btime);
}
totttime=tbm+totwtime;
printf("\n Total waiting time:%d",totwtime);
printf("\n Average waiting time:%f",(float)totwtime/n);
printf("\n Total turnaround time:%d",totttime);
printf("\n Avg turnaround time:%f",(float)totttime/n);


}
```

OUTPUT

```
Priority scheduling…..
Enter the number of processes : 5
Enter the burst time and priority
Process 1 : 10 3
Process 2 : 1 1
Process 3 : 2 4
Process 4 : 1 5
Process 5 : 5 2
Process Burst time Waiting Time Turn around time
2 1 0 1
5 5 1 6
1 10 6 16
3 2 16 18
4 1 18 19
Total waiting time : 28
Average waiting time : 7
Total turn around time : 52
Average turn around time :13
```

EXPERIMENT NO : 2

BANKER'S ALGORITHM

AIM

To implement the banker's algorithm for deadlock avoidance.

PROGRAM

```
#include<stdio.h>
int max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n,r;
void input();
void show();
void cal();
int main()
{
 int i,j;
 printf("********** Banker's Algorithm ************\n");
 input();
```

```
show();

cal();

return 0;

}


void input()

{

int i,j;

printf("Enter the no of Processes:\t");

scanf("%d",&n);

printf("\nEnter the no of resources instances:\t");

scanf("%d",&r);

printf("\nEnter the Max Matrix\n");

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

{

for(j=0;j<r;j++)

{

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

}


}

printf("\nEnter the Allocation Matrix\n");

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

{
```

```
for(j=0;j<r;j++)

{

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

}


}

printf("\nEnter the available Resources\n");

for(j=0;j<r;j++)

{

scanf("%d",&avail[j]);

}


}


void show()

{

int i,j;

printf("Process\t Allocation\t\t Max\t\t Available");

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

{

printf("\nP%d\t ",i+1);

for(j=0;j<r;j++)

{

printf("\t%d ",alloc[i][j]);
}

printf("\t");

for(j=0;j<r;j++)

{

printf("\t%d ",max[i][j]);

}

printf("\t");

if(i==0)

{

for(j=0;j<r;j++)

printf("%d ",avail[j]);

}


}


}


void cal()

{

int finish[100],temp,need[100][100],flag=1,k,c1=0;

int safe[100];

int i,j;

printf("\nSafe Sequence is:");

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

```
{
finish[i]=0;
}
//find need matrix
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
need[i][j]=max[i][j]-alloc[i][j];
}

}
printf("\n");
while(flag)
{
flag=0;
for(i=0;i<n;i++)
{
int c=0;
for(j=0;j<r;j++)
{
if((finish[i]==0)&&(need[i][j]<=avail[j]))
{
c++;
if(c==r)

{

for(k=0;k<r;k++)

{

avail[k]+=alloc[i][j];

finish[i]=1;

flag=1;

}

printf("P%d->",i+1);

if(finish[i]==1)

{

i=n;

}


}


}


}


}


}

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

```
{
if(finish[i]==1)
{
c1++;
}
else
{
printf("P%d->",i);

}

}
if(c1==n)
{
printf("\n The system is in safe state");

}
else
{
printf("\n Process are in dead lock");
printf("\n System is in unsafe state");
}

}
```

OUTPUT

```
Enter the no of Processes : 5

Enter the no of resources instances: 3

Enter the Max Matrix :

 7 5 3

 3 2 2

 9 0 2

 2 2 2

 4 3 3

Enter the Allocation Matrix :

 0 1 0

 2 0 0

 3 0 2

 2 1 1

 0 0 2

Enter the available Resources: 3 3 2

 Process Allocation Max Available

 P1 0 1 0 7 5 3 3 3 2

 P2 2 0 0 3 2 2

 P3 3 0 2 9 0 2

 P4 2 1 1 2 2 2

 P5 0 0 2 4 3 3

Safe Sequence is: P2-> P4 ->P5 ->P3 ->P1

The system is in safe state.
```

EXPERIMENT NO: 3

DISK SCHEDULING ALGORITHMS

AIM

To simulate the following disk scheduling algorithms.

a) FCFS b) SCAN c) C-SCAN

a) FCFS

PROGRAM

```
#include<stdio.h>
int main()
{
int i,j,sum=0,n;
int ar[20],tm[20];
int disk;
printf("FCFS Disk Scheduling....");
printf("\nEnter number of locations:\t");
scanf("%d",&n);
printf("\nEnter position of head:\t");
scanf("%d",&disk);
printf("\nEnter elements of disk queue:\n");
for(i=0;i<n;i++)
{
```

```
scanf("%d",&ar[i]);
tm[i]=disk-ar[i];
if(tm[i]<0)
{
tm[i]=ar[i]-disk;
}
disk=ar[i];
sum=sum+tm[i];
}
for(i=0;i<n;i++)
{
printf("%d +",tm[i]);
}
printf("\nMovement of total cylinders = %d",sum);
return 0;
}
```

OUTPUT

```
FCFS Disk Scheduling...
Enter number of locations: 8
Enter position of head: 53
Enter elements of disk queue: 98 183 37 122 14 124 65 67
45 + 85 + 146 + 85 + 108 + 110 + 59 + 2
Movement of total cylinders = 640
```

b) SCAN

PROGRAM

```
#include<stdio.h>

int main()

{

int i,j,sum=0,n;

int d[20];

int disk; //loc of head

int temp,max;

int dloc; //loc of disk in array

printf("SCAN Disk Scheduling....");

printf("\nEnter number of location\t");

scanf("%d",&n);

printf("\nEnter position of head\t");

scanf("%d",&disk);

printf("\nEnter elements of disk queue\n");

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

{
```

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

}

d[n]=disk;

n=n+1;

for(i=0;i<n;i++) // sorting disk locations

{

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

{

if(d[i]>d[j])

{

temp=d[i];

d[i]=d[j];

d[j]=temp;

}

}

}

max = d[n-1];

for(i=0;i<n;i++) // to find loc of disc in array

{

if(disk==d[i]) { dloc=i; break; }

}

for(i=dloc;i>=0;i--)

{

printf("%d -->",d[i]);
```

```
}
printf("0 -->");
for(i=dloc+1;i<n;i++)
{
printf("%d-->",d[i]);
}
sum = disk + max;
printf("\nMovement of total cylinders: %d",sum);
return 0;
}
```

OUTPUT

```
SCAN Disk Scheduling...
Enter number of locations: 8
Enter position of head: 53
Enter elements of disk queue: 98 183 37 122 14 124 65 67
53 37 14 0 65 67 98  122 124 183
Movement of total cylinders = 236
```

c) C-SCAN

PROGRAM

```
#include<stdio.h>
int main()
{
int i,j,sum=0,n;
int d[20];

int disk; //loc of head

int temp,max;

int dloc; //loc of disk in array

printf("C-SCAN Disk Scheduling....");

printf("\nEnter number of location\t");

scanf("%d",&n);

printf("\nEnter position of head\t");

scanf("%d",&disk);

printf("\nEnter elements of disk queue\n");

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

{

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

}

d[n]=disk;

n=n+1;

for(i=0;i<n;i++) // sorting disk locations

{

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

{

if(d[i]>d[j])

{

temp=d[i];

d[i]=d[j];
d[j]=temp;
}
}
}
max=199;
for(i=0;i<n;i++) // to find loc of disc in array
{
if(disk==d[i]) { dloc=i; break; }
}
for(i=dloc;i<n;i++)
{
printf("%d-->",d[i]);
}
printf("199 -->0 -->");
for(i=0;i<dloc;i++)
{
printf("%d -->",d[i]);
}
sum=d[i-1]+(max-disk)+max;
printf("\nmovement of total cylinders %d",sum);
return 0;
}
```

OUTPUT

```
C-SCAN Disk Scheduling...
```

```
Enter number of locations: 8

Enter position of head: 53

Enter elements of disk queue: 98 183 37 122 14 124 65 67

53 65 67 98  122 124 183 199 0 14 37

Movement of total cylinders = 382
```

EXPERIMENT NO: 4

PASS ONE OF TWO PASS ASSEMBLER

AIM

To implement pass one of a two pass assembler

PROGRAM

```
#include<stdio.h>

#include<string.h>

void main()

{

FILE *f1,*f2,*f3,*f4;

char s[100],lab[30],opcode[30],opa[30],opcode1[30],opa1[30];

int locctr,x=0;

f1=fopen("input.txt","r");

f2=fopen("opcode.txt","r");

f3=fopen("out1.txt","w");

f4=fopen("sym1.txt","w");

while(fscanf(f1,"%s%s%s",lab,opcode,opa)!=EOF)

{

                    if(strcmp(lab,"**")==0)
```

```
{
if(strcmp(opcode,"START")==0)
{
    fprintf(f3,"%s %s %s",lab,opcode,opa);
    locctr=(atoi(opa));

}
else
{
    rewind(f2);
    x=0;
    while(fscanf(f2,"%s%s",opcode1,opa1)!=EOF)
    {
    if(strcmp(opcode,opcode1)==0)
    {
    x=1;
    }
    }
    if(x==1)
    {
    fprintf(f3,"\n %d %s %s %s",locctr,lab,opcode,opa);
    locctr=locctr+3;
    }
}
```

```
}
else
{
if(strcmp(opcode,"RESW")==0)
{
fprintf(f3,"\n %d %s %s %s",locctr,lab,opcode,opa);
fprintf(f4,"\n %d %s",locctr,lab);
locctr=locctr+(3*(atoi(opa)));
}
else if(strcmp(opcode,"WORD")==0)
{
fprintf(f3,"\n %d %s %s %s",locctr,lab,opcode,opa);
fprintf(f4,"\n %d %s",locctr,lab);
locctr=locctr+3;
}
else if(strcmp(opcode,"BYTE")==0)
{
fprintf(f3,"\n %d %s %s %s",locctr,lab,opcode,opa);
fprintf(f4,"\n %d %s",locctr,lab);
locctr=locctr+1;
}
else if(strcmp(opcode,"RESB")==0)
{
fprintf(f3,"\n %d %s %s %s",locctr,lab,opcode,opa);
```

```
                fprintf(f4,"\n %d %s",locctr,lab);
                locctr=locctr+1;
                }
                else
                {
                fprintf(f3,"\n %d %s %s %s",locctr,lab,opcode,opa);
                fprintf(f4,"\n %d %s",locctr,lab);
                locctr=locctr+(atoi(opa));
                }
                }
}
}
```

INPUT FILES

input.txt

```
** START 2000
** LDA FIVE
** STA ALPHA
** LDCH CHARZ
** STCH C1
ALPHA RESW 1
```

```
FIVE WORD 5
CHARZ BYTE C'Z'
C1 RESB 1
** END **
```

opcode.txt

```
START *
LDA 03
STA 0F
LDCH 53
STCH 57
END
```

OUTPUT FILES

out1.txt

```
** START 2000
 2000 ** LDA FIVE
 2003 ** STA ALPHA
 2006 ** LDCH CHARZ
 2009 ** STCH C1
 2012 ALPHA RESW 1
 2015 FIVE WORD 5
 2018 CHARZ BYTE C'Z'
 2019 C1 RESB 1
 2020 ** END **
```

sym1.txt

```
 2012 ALPHA
 2015 FIVE
 2018 CHARZ
 2019 C1
```

EXPERIMENT NO: 5

PASS TWO OF TWO PASS ASSEMBLER

AIM

To implement pass two of a two pass assembler

PROGRAM

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>

void main()

   {

char opcode[20],operand[20],symbol[20],label[20],code[20],mnemonic[25],
character, add[20],objectcode[20];

int flag,flag1,locctr,location,loc;

FILE *fp1,*fp2,*fp3,*fp4;



fp1=fopen("out3.txt","r"); fp2=fopen("twoout.txt","w");

fp3=fopen("opcode.txt","r"); fp4=fopen("sym1.txt","r");

fscanf(fp1,"%s%s%s",label,opcode,operand);

if(strcmp(opcode,"START")==0)

{ fprintf(fp2,"%s\t%s\t%s\n",label,opcode,operand);

fscanf(fp1,"%d%s%s%s",&locctr,label,opcode,operand);
```

```
}
while(strcmp(opcode,"END")!=0)
{ flag=0;
fscanf(fp3,"%s%s",code,mnemonic);
while(strcmp(code,"END")!=0)
{ if((strcmp(opcode,code)==0) && (strcmp(mnemonic,"*"))!=0)
{ flag=1;
break;
}
fscanf(fp3,"%s%s",code,mnemonic);



}
if(flag==1)
{ flag1=0; rewind(fp4);
while(!feof(fp4))
{
fscanf(fp4,"%s%d",symbol,&loc);
if(strcmp(symbol,operand)==0)
{
flag1=1; break;
} }
if(flag1==1)
{
```

```
sprintf(add,"%d",loc);
strcpy(objectcode,strcat(mnemonic,add));
} }
else if(strcmp(opcode,"BYTE")==0 || strcmp(opcode,"WORD")==0)
{
if((operand[0]=='C') || (operand[0]=='X'))
{
character=operand[2];
sprintf(add,"%d",character);
strcpy(objectcode,add);
}
else
{
strcpy(objectcode,add);
} }
else
strcpy(objectcode,"\0");
fprintf(fp2,"%s\t%s\t%s\t%d\t%s\n",label,opcode,operand,locctr,objectcode);
fscanf(fp1,"%d%s%s%s",&locctr,label,opcode,operand);
}
fprintf(fp2,"%s\t%s\t%s\t%d\n",label,opcode,operand,locctr);
fclose(fp1);
fclose(fp2);
fclose(fp3);
```

```
 fclose(fp4);
}
```

INPUT FILES

opcode.txt

```
START *
LDA 03
STA 0F
LDCH 53
STCH 57
END +
```

out3.txt

```
** START 2000
2000 ** LDA FIVE
2003 ** STA ALPHA
2006 ** LDCH CHARZ
2009 ** STCH C1
2012 ALPHA RESW 1
2015 FIVE WORD 5
2018 CHARZ BYTE C'Z'
2019 C1 RESB 1
2020 ** END **
```

sym1.txt

```
 2012 ALPHA
 2015 FIVE
 2018 CHARZ
 2019 C1
```

OUTPUT FILES

twoout.txt

```
**      START       2000
**      LDA   FIVE  2000    032018
**      STA   ALPHA         2003    0F2015
**      LDCH        CHARZ         2006    532019
**      STCHC1      2009    572019
ALPHA         RESW                  1       2012
FIVE WORD          5       2015    2019
CHARZ        BYTE         C'Z'    2018    90
C1      RESB1       2019
**      END  **     2020
```

EXPERIMENT NO: 6

TWO PASS MACRO PROCESSOR

AIM

To implement a two pass macro processor.

PROGRAM

Pass one of two pass macro processor

```
#include<stdio.h>

#include<string.h>

void main()

{


        char macros[20][10], label[20],opcode[20],operand[20];

        int i, j, n,m=0;

        FILE *fp1, *fp[10];


        fp1=fopen("inputm.txt","r");

        fscanf(fp1,"%s%s%s",label,opcode,operand);

        while(strcmp(opcode,"END")!=0)

        {
```

```
        if(!strcmp(opcode,"MACRO")){
            fp[m]=fopen(operand,"w");
            m++;
            fscanf(fp1,"%s%s%s",label,opcode,operand);
            while(strcmp(opcode,"MEND")!=0){
                fprintf(fp[m-1],"%s\t%s\t%s\n",label,opcode,operand);
                fscanf(fp1,"%s%s%s",label,opcode,operand);
            }
        }
        fscanf(fp1,"%s%s%s",label,opcode,operand);
    }
}
```

INPUT FILES

inputm.txt

```
** MACRO m1
** LDA ALPHA
** STA BETA
** MEND **
** MACRO m2
** MOV a,b
** MEND **
** START 1000
** LDA a
** CALL m1

** CALL m2

** END **
```

OUTPUT FILES

m1.txt

```
**      LDA  ALPHA

**      STA   BETA
```

m2.txt

```
**      MOV a,b
```

<u>Pass two of two pass assemblers</u>

PROGRAM

```
#include<stdio.h>

#include<string.h>

void main()

{


        char macros[20][10], label[20],opcode[20],operand[20];

        int i, j, n,m=0;

        FILE *fp1, *fp[10],*fp2;
```

```
fp1=fopen("inputm.txt","r");

fp2=fopen("macro_out.txt","w");

fscanf(fp1,"%s%s%s",label,opcode,operand);

while(strcmp(opcode,"END")!=0)

 {

   if(!strcmp(opcode,"CALL"))

 {

        fp[m]=fopen(operand,"r");

            m++;

        fscanf(fp[m-1],"%s%s%s",label,opcode,operand);

         while(!feof(fp[m-1]))

         {

           fprintf(fp2,"%s\t%s\t%s\n",label,opcode,operand);

         fscanf(fp[m-1],"%s%s%s",label,opcode,operand);

         }

 }

 else

 {

     fprintf(fp2,"%s\t%s\t%s\n",label,opcode,operand);

 }
```

```
            fscanf(fp1,"%s%s%s",label,opcode,operand);
            }
        fprintf(fp2,"%s\t%s\t%s\n",label,opcode,operand);
}
```

INPUT FILES

inputm.txt

```
** MACRO m1
** LDA ALPHA
** STA BETA
** MEND **
** MACRO m2
** MOV a,b
** MEND **
** START 1000
** LDA a
** CALL m1
** CALL m2
** END **
```

OUTPUT FILES

m1.txt

```
**     LDA  ALPHA

**     STA   BETA
```

m2.txt

```
**     MOV   a,b
```

output file

```
**     MACRO              m1

**     LDA  ALPHA

**     STA   BETA

**     MEND **

**     MACRO              m2

**     MOV   a,b

**     MEND **

**     START        1000

**     LDA  a

**     END  ***
```

EXPERIMENT NO: 7

ONE PASS MACRO PROCESSOR

AIM

To implement a single pass macro processor.

PROGRAM

```
#include<stdio.h>

#include<conio.h>

#include<ctype.h>

#include<string.h>

int m=0,i,j,flag=0;

char c,*s1,*s2,*s3,*s4,str[50]=" ",str1[50]=" ";

char mac[10][10];

void main()

{

FILE *fpm=fopen("macro.txt","r");

FILE *fpi=fopen("minput.txt","r");

FILE *fpo=fopen("moutput.txt","w");

clrscr();

while(!feof(fpm))

{

fgets(str,50,fpm);
```

```
s1=strtok(str," ");
s2=strtok(NULL," ");
if(strcmp(s1,"MACRO")==0)
{
strcpy(mac[m],s2);
m++;
}
s1=s2=NULL;
}
fgets(str,50,fpi);
while(!feof(fpi))
{
flag=0;
strcpy(str1,str);
for(i=0;i<m;i++)
{
if(strcmp(str1,mac[i])==0)
{
rewind(fpm);
while(!feof(fpm))
{
fgets(str,50,fpm);
s2=strtok(str," ");
s3=strtok(NULL," ");
if(strcmp(s2,"MACRO")==0&&strcmp(s3,str1)==0)

{

fgets(str,50,fpm);

strncpy(s4,str,4);

s4[4]='\0';

while(strcmp(s4,"MEND")!=0)

{

fprintf(fpo,"%s",str);

printf("\n####%s",str);

fgets(str,50,fpm);

strncpy(s4,str,4);

s4[4]='\0';

}

}

}

flag=1;

break;

}

}

if(flag==0)

{

fprintf(fpo,"%s",str);

printf("%s",str);

}
```

```
fgets(str,50,fpi);

}

fclose(fpm);

fclose(fpi);

fclose(fpo);

}
```

INPUT FILES

Macro.txt

```
MACRO ADD1

MOV A,B

ADD C

MEND

MACRO SUB1

STORE C

MEND
```

MInput.txt

```
MOV B,10

MOV C,20

ADD1
```

```
MUL C

SUB1

END
```

OUTPUT

MOutput.txt

```
MOV B,10

MOV C,20

MOV A,B

ADD C

MUL C

STORE C

END
```

EXPERIMENT NO: 8

ABSOLUTE LOADER

AIM

To implement an Absolute Loader.

PROGRAM

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void main()
{
        FILE *fp;
        int i,addr1,l,j,staddr1;
        char name[10],line[50],name1[10],addr[10],rec[10],ch,staddr[10];

          printf("enter program name:" );
         scanf("%s",name);
          fp=fopen("objectcode.txt","r");
           fscanf(fp,"%s",line);
        for(i=2,j=0;i<8,j<6;i++,j++)
  name1[j]=line[i];
  name1[j]='\0';
  printf("name from obj. %s\n",name1);
 if(strcmp(name,name1)==0)
 {
      fscanf(fp,"%s",line);
     do
     {

             if(line[0]=='T')
            {
                   for(i=2,j=0;i<8,j<6;i++,j++)
                 staddr[j]=line[i];
                staddr[j]='\0';
               staddr1=atoi(staddr);
                i=12;
            while(line[i]!='$')
           {
                if(line[i]!='^')
               {
                     printf("00%d \t %c%c\n", staddr1,line[i],line[i+1]);
                     staddr1++;
                      i=i+2;
```

```
              }
           else i++;
           }
    }
   else if(line[0]='E')
           printf("jump to execution address:%s",&line[2]);
       fscanf(fp,"%s",line);
   }while(!feof(fp) );



  }
   fclose(fp);
}
```

objectcode.txt

```
H^SAMPLE^001000^0035
T^001000^0C^001003^071009$
T^002000^03^111111$
H^SAMPLE^001000^0035
T^001000^0C^001003^071009$
T^002000^03^111111$
E^001000
```

OUTPUT

```
enter program name:SAMPLE

name from obj. SAMPLE

001000   00

001001   10

001002   03

001003   07

001004   10

001005   09

002000   11

002001   11

002002   11

jump to execution address:001000
```