2.a Write a shell script to compare two strings.

```
echo “Enter two strings”
read a
read b
if [ $a = $b ]
then
echo “Strings are equal strings Matched”
else
echo “Strings are not equal strings not match”
fi
```

2.b Write a shell script to extract the first and last character from a string.

```
a="abcdef"
first="${a:0:1}"
last="${a: -1}"
echo "$first"
echo "$last"
```

2.c Write a shell script to find whether the given number is palindrome or not.

```
echo "Enter the number : "
read n
num=0
a=$n
while [ $n -gt 0 ]
do
num=`expr $num \* 10^$k`
k=`expr $n % 10`
num=`expr $num + $k`
n=`expr $n / 10`
done
if [ $num -eq $a ]
then
echo "Its a Palindrome"
else
echo "Not a Palindrome"
fi
```

3.a Write a shell script to find the factorial of a given number using for loop.

```
echo "Enter the number : "
read n
fact=1
for I in $(seq 1 $n);
do
fact=$((fact*i))
done
echo "Factorial = $fact"
```

3.b Write a shell script to find the sum of n numbers using while loop.

```
echo "Enter the size : "
read n
i=1
sum=0
echo "Enter the numbers"
while [ $i -le $n ]
do
read num
sum=$((sum + num))
i=$((i + 1))
done
echo "sum = $sum"
```

3.c Write a shell script to implement menu driven program to perform all arithmetic operation using case statement.

```
echo "Enter two numbers:"
read a
read b
echo "MENU 1. Addition 2. Subtraction 3. Multiplication 4. Division"
echo "Enter the choice:"
read c
case $c in
   1) echo "Sum=$(expr $a + $b)";;
   2) echo "Subtraction=$(expr $a - $b)";;
   3) echo "Multiplication=$(expr $a \* $b)";;
   4) echo "Division=$(expr $a / $b)";;
   *) echo "Invalid Choice: Try Again";;
esac
```

3.d Write a shell script to print following pattern.

```
*
* *
* * *
* * * *
```

```
echo “ enter number of rows”
read n i=1 j=1
while [ $i –le $n ]
do
while [ $j –le $i ]
do
echo –n “*”
j=$((j + 1))
done
echo
i=$((i + 1))
done
```

4.a Converting File names from Uppercase to Lowercase

```
a="THIS IS A TEST"
echo "$a" | tr '[:upper:]' '[:lower:]'
b="sathyabama"
echo "$b" | tr '[:lower:]' '[:upper:]'
```

4.b Write a shell script to count the number of characters, words and lines in the file.

Create a file :

```
cat > a
My university name is Sathyabama
My department is Computer Science
This is Chennai

echo "enter file name"
read file
c=`cat $file | wc -c`
w=`cat $file | wc -w`
l=`grep -c "." $file`
echo "Number of characters is $c"
echo "Number of words is $w"
echo "Number of Lines is $l"
```

4.c Write a shell script to read and check the file exists or not, if not create the file.

```
echo “enter name of file”
read filename
if [ -f $filename ]
then
echo “File $filename Exits!”
else
touch $filename
fi
```

5. Manipulate Date/Time/Calendar

```
echo "Date"
echo "Today is $(date +'%m/%d/%y')"
echo "Calendar"
echo $(cal 9 2024)
echo "Time"
echo $(date +"%I:%S:%M")
```

6. Showing various system information

```
echo "SYSTEM INFORMATION"
echo “Hello ,$LOGNAME”
echo “Current Date is = $(date)”
echo “User is ‘who I am’”
echo “Current Directory = $(pwd)”
echo "Network Name and Node Name = $(uname -n)"
echo "Kernal Name =$(uname -s)"
echo "Kernal Version=$(uname -v)"
echo "Kernal Release =$(uname -r)"
echo "Kernal OS =$(uname -o)"
echo “Proessor Type = $(uname -p)”
echo “Kernel Machine Information = $(uname –m)”
echo "All Information =$(uname -a)"
```

7. Implementation of process scheduling mechanism – FCFS, SJF, Priority Queue.

FCFS

```
#include<stdio.h>
#include<conio.h>
void main()
{ int nop,wt[10],twt,tat[10],ttat,i,j,bt[10],t;
float awt,atat;
awt=0.0;
atat=0.0;
printf("Enter the no.of process:");
scanf("%d",&nop);
for(i=0;i<nop;i++)
{ printf("Enter the burst time for process %d: ", i);
scanf("%d",&bt[i]); }
wt[0]=0;
tat[0]=bt[0];
twt=wt[0];
ttat=tat[0];
for(i=1;i<nop;i++)
{ wt[i]=wt[i-1]+bt[i-1];
tat[i]=wt[i]+bt[i];
twt+=wt[i];
ttat+=tat[i]; }
awt=(float)twt/nop;
atat=(float)ttat/nop;
printf("\nProcessid\tBurstTime\tWaitingTime\tTurnaroundTime\n");
for(i=0;i<nop;i++)
printf("%d\t\t%d\t\t%d\t\t%d\n",i,bt[i],wt[i],tat[i]);
printf("\nTotal Waiting Time:%d\n",twt);
printf("\nTotal Around Time:%d\n",ttat);
printf("\nAverage Waiting Time:%f\n",awt);
printf("\nAverage Total Around Time:%f\n",atat); }
```

SJF

```
#include<stdio.h>
#include<conio.h>
void main()
{ int nop,wt[10],twt,tat[10],ttat,i,j,bt[10],t;
float awt,atat;
clrscr();
awt=0.0;
atat=0.0;
printf("Enter the no.of process:");
scanf("%d",&nop);
for(i=0;i<nop;i++)
{ printf("Enter the burst time for process %d: ", i);
scanf("%d",&bt[i]); }
for(i=0;i<nop;i++)
{ for(j=i+1;j<nop;j++)
{ if(bt[i]>=bt[j])
{ t=bt[i];
bt[i]=bt[j];
bt[j]=t; } } }
wt[0]=0;
tat[0]=bt[0];
twt=wt[0];
ttat=tat[0];
for(i=1;i<nop;i++)
{ wt[i]=wt[i-1]+bt[i-1];
tat[i]=wt[i]+bt[i];
twt+=wt[i];
ttat+=tat[i]; }
awt=(float)twt/nop;
atat=(float)ttat/nop;
printf("\nProcessid\tBurstTime\tWaitingTime\tTurnaroundTime\n");
for(i=0;i<nop;i++)
printf("%d\t\t%d\t\t%d\t\t%d\n",i,bt[i],wt[i],tat[i]);
printf("\nTotal Waiting Time:%d\n",twt);
printf("\nTotal Around Time:%d\n",ttat);
printf("\nAverage Waiting Time:%f\n",awt);
printf("\nAverage Total Around Time:%f\n",atat); }
```

PRIORITY

```
#include<stdio.h>
#include<conio.h>
void main()
{ int nop,t,wt[10],twt,tat[10],ttat,i,j,p[10],b[10],tmp;
float awt, atat;
clrscr();
awt=0.0;
atat=0.0;0
printf("Enter the number of process:");
scanf("%d",&nop);
for(i=0;i<nop;i++)
{ printf("Enter the burst time of Process %d:",i);
scanf("%d",&b[i]); }
for(i=0;i<nop;i++)
{ printf("Enter the priority number of each Process %d:",i);
scanf("%d",&p[i]); }
for(i=0;i<nop;i++)
{ for(j=i+1;j<nop;j++)
{ if(p[i]>p[j])
{ t=p[i];
p[i]=p[j];
p[j]=t;
tmp=b[i];
b[i]=b[j];
b[j]=tmp; } } }
wt[0]=0;
tat[0]=b[0];
twt=wt[0];
ttat=tat[0];
for(i=1;i<nop;i++)
{ wt[i]=wt[i-1]+b[i-1];
tat[i]=wt[i]+b[i];
twt+=wt[i];
ttat+=tat[i]; }
awt=(float)twt/nop;
atat=(float)ttat/nop;
printf("Process No:\tPriority:\tBurst Time:\tWaiting Time\tTurnaround Time:\n");
for(i=0;i<nop;i++)
printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",i,p[i],b[i],wt[i],tat[i]);
printf("Total TurnAround Time:%d\n",ttat);
printf("Total Waiting Time:%d\n",twt);
printf("Average Waiting Time:%f\n",awt);
printf("Average Turnaround Time:%f\n",atat); }
```

8. Implement Reader – Writer Problem.

```
#include<stdio.h>
#include <stdlib.h>
void main()
{typedef int semaphore;
semaphore sread=0, swrite=0;
int ch,r=0;
printf("\nReader writer");
do
{printf("\nMenu");
printf("\n1.Read from file");
printf("\n2.Write to file");
printf("\n3.Exit the reader");
printf("\n4.Exit the writer");
printf("\n5.Exit");
printf("\nEnter your choice:");
scanf("%d",&ch);
switch(ch)
{case 1: if(swrite==0)
{sread=1;
r+=1;
printf("\nReader %d reads",r);}
break;
case 2: if(sread==0 && swrite==0)
{swrite=1;
printf("\nWriter in Progress");}
break;
case 3: if(r!=0)
{printf("\nThe reader %d closes the file",r);
r-=1;}
break;
case 4: if (swrite==1)
{printf("\nWriter close the file");
swrite=0;}
break;
case 5:
exit(0);}}
while(ch<6);}
```

9. Implement Dinner’s Philosopher Problem.

```
#include<stdio.h>
#define L (i+4) %5
#define R (i+1) %5
#define T 0
#define H 1
#define E 2
int s[5];
void pf(int);
void t(int);
void tf(int);
void p(int i)
{if(s[i]==0)
{tf(i);
if(s[i]==E)
printf("\n Eating in process....");
pf(i);}}
void pf(int i)
{s[i]=T;
printf("\n p %d completed its works",i);
t(L);
t(R);}
void tf(int i)
{s[i]=H;
t(i);}
void t(int i)
{if(s[i]==H && s[L]!=E && s[R]!=E)
{printf("\n p %d can eat",i);
s[i]=E;}}
void main()
{int i;
for(i=1;i<=5;i++)
s[i]=0;
printf("\n\t\t\t Dining Philosopher Problem");
for(i=1;i<=5;i++)
{printf("\n\n the p %d falls hungry\n",i);
p(i);}}
```

10. Implement First Fit, Worst Fit, Best Fit allocation strategy.

```
#include <stdio.h>
int main()
{int ms, bod, sb[20], sp[20], z[20];
int f[20], f2[20], f3[20], r[20], r1[20];
int np, i, j, k, s, l= 0;
printf("Enter memory size: ");
scanf("%d", &ms);
printf("Enter number of blocks: ");
scanf("%d", &bod);
printf("Enter size of each block:\n");
for (i = 1; i <= bod; i++)
{printf("Block[%d]: ", i);
scanf("%d", &sb[i]);
f[i] = f2[i] = f3[i] = r[i] = r1[i] = z[i] = sb[i];}
printf("Enter number of processes: ");
scanf("%d", &np);
printf("Enter size of each process:\n");
for (i = 1; i <= np; i++)
{printf("Process[%d]: ", i);
scanf("%d", &sp[i]);}
printf("\nFIRST FIT\n");
for (i = 1; i <= np; i++)
{for (j = 1; j <= bod; j++)
{if (sb[j] >= sp[i] && f[j] != 0)
{printf("Process p[%d] is allocated to Block[%d]\n", i, j);
f[j] = 0;
z[j] = sb[j] - sp[i];
s++;
break;}}
if (f[j] != 0)
{printf("Process p[%d] cannot be allocated\n", i);}}
printf("\nRemaining space left in each block:\n");
for (i = 1; i <= bod; i++)
{printf("Block[%d]: free space = %d\n", i, z[i]); }
printf("\nUnallocated Blocks:\n");
for (i = 1; i <= bod; i++)
{if (f[i] != 0)
{printf("Block [%d] unallocated\n", i);}}
if (s == bod)
{printf("\n No Block is left unallocated\n");}
printf("\nBEST FIT\n");
for (i = 2; i <= bod; i++)
{for (j = 1; j < i; j++)
{if (sb[i] >= sb[j])
{r[i]++;}
else
{r[j]++;} }}
for (i = 1; i <= bod; i++)
{sb[r[i]] = sb[i];
z[r[i]] = sb[i];}
for (i = 1; i <= np; i++)
{for (j = 1; j <= bod; j++)
{if (sb[j] >= sp[i] && f2[j] != 0)
{for (k = 1; k <= bod; k++)
{if (r[k] == j)
{l = k;}}
printf("Process p[%d] is allocated to Block[%d]\n", i, l);
f2[j] = 0;
z[j] = sb[j] - sp[i];
s++;
break;}}
if (f[j] != 0)
{printf("Process p[%d] cannot be allocated\n", i);}}
printf("\n free space in each block:\n");
for (i = 1; i <= bod; i++)
{printf("Block [%d]: free space = %d\n", i, z[r[i]]);}
printf("\nUnallocated Blocks:\n");
for (i = 1; i <= bod; i++)
{if (f2[r[i]] != 0) {
printf("Block [%d] unallocated\n", i);}}
if (s == bod)
{printf("\n No Block is left unallocated\n");}
printf("\nWORST FIT\n");
for (i = 2; i <= bod; i++)
{for (j = 1; j < i; j++)
{if (sb[i] <= sb[j])
{r1[i]++;}
else
{r1[j]++;
}}}
for (i = 1; i <= bod; i++)
{sb[r1[i]] = sb[i];
z[r1[i]] = sb[i];}
for (i = 1; i <= np; i++)
{for (j = 1; j <= bod; j++)
{if (sb[j] >= sp[i] && f3[j] != 0) {
for (k = 1; k <= bod; k++)
{if (r1[k] == j)
{l = k;}}
printf("Process p[%d] is allocated to Block[%d]\n", i, l);
f3[j] = 0;
z[j] = sb[j] - sp[i];
s++;
break;}}
```

```
if (f[j] != 0)
{printf("Process p[%d] cannot be allocated\n", i); }}
printf("\n free space in each block:\n");
for (i = 1; i <= bod; i++) {
printf("Block [%d]: free space = %d\n", i, z[r1[i]]);}
printf("\nUnallocated Blocks:\n");
for (i = 1; i <= bod; i++)
{if (f3[r1[i]] != 0)
{printf("Block [%d] unallocated\n", i);}}
if (s == bod) {
printf("\n No Block is left unallocated\n");}}
```

OUTPUT:

```
Enter memory size: 1000
Enter number of blocks: 5
Enter size of each block:
Block[1]: 300
Block[2]: 400
Block[3]: 200
Block[4]: 700
Block[5]: 600

Enter number of processes: 4
Enter size of each process:
Process[1]: 212
Process[2]: 417
Process[3]: 112
Process[4]: 426

FIRST FIT
Process p[1] is allocated to Block[3]
Process p[2] is allocated to Block[2]
Process p[3] is allocated to Block[1]
Process p[4] cannot be allocated

Remaining space left in each block:
Block[1]: free space = 88
Block[2]: free space = 0
Block[3]: free space = 0
Block[4]: free space = 700
Block[5]: free space = 600

Unallocated Blocks:
Block [4] unallocated
Block [5] unallocated
```

```
BEST FIT
Process p[1] is allocated to Block[3]
Process p[2] is allocated to Block[2]
Process p[3] is allocated to Block[1]
Process p[4] cannot be allocated

free space in each block:
Block [1]: free space = 88
Block [2]: free space = 0
Block [3]: free space = 0
Block [4]: free space = 700
Block [5]: free space = 600

Unallocated Blocks:
Block [4] unallocated
Block [5] unallocated

WORST FIT
Process p[1] is allocated to Block[4]
Process p[2] is allocated to Block[5]
Process p[3] cannot be allocated
Process p[4] cannot be allocated

free space in each block:
Block [1]: free space = 300
Block [2]: free space = 400
Block [3]: free space = 200
Block [4]: free space = 0
Block [5]: free space = 174

Unallocated Blocks:
Block [3] unallocated

No Block is left unallocated
```

11. Implement Bankers Algorithm

```
#include <stdio.h>
#define M 10
int np,nr,r[M],all[M][M],max[M][M],need[M][M],ava[M],f[M],safe[M];
void ir() {
printf("Enter number of resources: ");
scanf("%d",&nr);
printf("Enter the resource instances: ");
for(int i=0;i<nr;i++)
scanf("%d",&r[i]);}
void ip() {
printf("Enter number of processes: ");
scanf("%d",&np);
printf("Enter allocated resources for each process:\n");
for(int i=0;i<np;i++)
for(int j=0;j<nr;j++)
scanf("%d",&all[i][j]);
printf("Enter maximum resource requirements for each process:\n");
for(int i=0;i<np;i++)
for(int j=0;j<nr;j++)
scanf("%d",&max[i][j]);
printf("Available resources:\n");
for(int i=0;i<nr;i++) {
ava[i]=r[i];
for(int j=0;j<np;j++)
ava[i]-=all[j][i];
printf("%d ",ava[i]);}
printf("\n");}
void cn() {
printf("Need matrix:\n");
for(int i=0;i<np;i++) {
for(int j=0;j<nr;j++) {
need[i][j]=max[i][j]-all[i][j];
printf("%d ",need[i][j]);}
printf("\n");}}
void sc() {
int work[M],finish[M]={0},count=0;
for(int i=0;i<nr;i++)
work[i]=ava[i];
while(count<np) {
int found=0;
for(int i=0;i<np;i++) {
if(!finish[i]) {
int j;
for(j=0;j<nr;j++)
if(need[i][j]>work[j])
break;
if(j==nr) {
for(int k=0;k<nr;k++)
work[k]+=all[i][k];
safe[count++]=i;
finish[i]=1;
found=1;}}}
if(!found) {
printf("No safe sequence exists.\n");
return;}}
printf("Safe sequence: ");
for(int i=0;i<np;i++)
printf("P%d ",safe[i]);
printf("\n");}
int main() {
ir();
ip();
cn();
sc();
return 0;}
```

OUTPUT:

```
Enter number of resources: 3
Enter the resource instances: 10 5 7
Enter number of processes: 5
Enter allocated resources for each process:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Enter maximum resource requirements for each process:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Available resources:
3 5 3

Need matrix:
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1
Safe sequence: P1 P3 P4 P0 P2
```

12. Implement the producer consumer problem

```
#include<stdio.h>
#include <stdlib.h>
int main()
{int s,n,b=0,p=0,c=0;
printf("\nProducer and Consumer Problem");
do
{printf("\nMenu");
printf("\n1.Produce an item");
printf("\n2.Consume an item");
printf("\n3.Add item to the buffer");
printf("\n4.Display status");
printf("\n5.Exit");
printf("\nEnter the choice:");
scanf("%d",&s);
switch(s)
{case 1:
p=p+1;
printf("\nItem Produced");
break;
case 2:
{c=c+1;
b=b-1;
printf("\nItem Consumed");}
break;
case 3:
{b=b+1;
printf("\nItem added to buffer");}
break;
case 4:
printf("No.of items produced :%d",p);
printf("\n No.of consumed items:%d",c);
printf("\n No.of buffered item:%d",b);
break;
case 5:
exit(0);}}
while(s<=5);
return 0;}
```

13. Implement FIFO PAGE replacement algorithm.

```
#include<stdio.h>
void main()
{int is[]={7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1};
int pf=0;
int f=3;
int p=sizeof(is)/sizeof(is[0]);
int t[3]={-1,-1,-1};
printf("Inc\t\tF1\t\tF2\t\tF3\n");
for(int m=0;m<p;m++)
{int fd=0;
for(int n=0;n<f;n++)
{if(is[m]==t[n])
{fd=1;
break;}}
if(!fd)
{pf++;
int ri=-1;
for(int n=0;n<f;n++)
{if(t[n]==-1)
{ri=n;
break;}}
if(ri==-1)
{ri=(pf-1)%f;}
t[ri]=is[m];}
printf("%d\t\t",is[m]);
for(int n=0;n<f;n++)
{if(t[n]!=-1)
printf("%d\t\t",t[n]);
else
printf("-\t\t");}
printf("\n");}
printf("Total Page Faults: %d\n",pf);}
```