}

8. Given n processes with their burst times and arrival times, write a program to find average waiting time and average turn-around time using FCFS scheduling algorithm.

```
#include <stdio.h>
void sort(int a[][2],int n)
{
for(int i=0;i<n;i++)
{
for(int j=i+1;j<n;j++)
{
if(a[i][0]>a[j][0])
{
int t=a[i][0];
int k=a[i][1];
a[i][0]=a[j][0];
a[i][1]=a[j][1];
a[j][0]=t;
a[j][1]=k;
}
}
}
}
```

```
int main(){

int n;

printf("Enter the no. of process");

scanf("%d",&n);

int arr[n][2];

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

{

printf("\n enter the arrival and burst time for the %dth process\t",i+1);

scanf("%d%d",&arr[i][0],&arr[i][1]);

}

sort(arr,n);

int wait[n],tat[n],averageWait=0,averageTAT=0;

wait[0]=0;

for(int i=1;i<n;i++)

    {

        wait[i]=wait[i-1]+arr[i-1][1];

        averageWait+=wait[i];

    }


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

    {

        tat[i]=wait[i]+arr[i][1];

        averageTAT+=tat[i];
```

```
      }
double a=averageTAT/(1.0*n), b=averageWait/(1.0*n);
printf("Process no.\t Arrival time \t Burst Time\t Waiting Time \t Turn around Time\n");
for(int i=0;i<n;i++)
      {
           printf("%d\t\t %d\t\t %d \t\t %d \t\t%d\n",i+1,arr[i][0],arr[i][1],wait[i],tat[i]);
      }
printf("The average wait time is: %lf and the average turn around time is: %lf",b,a);
return 0;
}
```

9. Given n processes with their burst and arrival times, write a program to find average waiting time and average turn-around time using shortest job first scheduling algorithm.

```
#include <stdio.h>

int main()

{

int n;

printf("Enter the no. of process");

scanf("%d",&n);

int arr[n][2],burst[n];

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

     {

        printf("\n enter the arrival and burst time for the %dth process\t",i+1);

         scanf("%d%d",&arr[i][0],&arr[i][1]);

        burst[i]=arr[i][1];

     }

int wait[n],tat[n],prev[n];

int t=0,averageTAT=0,averageWait=0;

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

     {

       wait[i]=0;
```

```
        prev[i]=0;
      }
while(1)
      {
        int ind=0,mini=100000;
for(int i=0;i<n;i++)
        {
            if(t>=arr[i][0]&&mini>arr[i][1]&&arr[i][1]>0)
         {
                   mini=arr[i][1];
                   ind=i;
         }
        }
if(mini==100000)
     break;
arr[ind][1]-=1;
wait[ind]+=t-prev[ind];
 t++;
prev[ind]=t;
}
for(int i=0;i<n;i++)
```

```
      {
         tat[i]=wait[i]+burst[i];
         averageTAT+=tat[i];
         averageWait+=wait[i];
      }
double a=averageTAT/(1.0*n), b=averageWait/(1.0*n);
printf("Process no.\t Arrival time \t Burst Time\t Waiting Time \t Turn around
Time\n");
for(int i=0;i<n;i++)
      {
          printf("%d\t\t %d\t\t %d \t\t %d
\t\t%d\n",i+1,arr[i][0],arr[i][1],wait[i],tat[i]);
      }
printf("The average wait time is: %lf and the average turn around time is:
%lf",b,a);
return 0;
}
```

10. Given n processes with their burst and arrival times along with their priorities, write a program to find average waiting time and average

turn-around time using preemptive and non-preemptive versions of priority scheduling.

PREEMPTIVE

```
#include <stdio.h>
int main()
{
int n;
printf("Enter the no. of process");
scanf("%d",&n);
int arr[n][3],burst[n];
for(int i=0;i<n;i++)
    {
        printf("\n enter the arrival,burst time and priority number for the %dth
process\t",i+1);
        scanf("%d%d%d",&arr[i][0],&arr[i][1],&arr[i][2]);
        burst[i]=arr[i][1];
    }
int wait[n],tat[n],prev[n];
int t=0,averageTAT=0,averageWait=0;
for(int i=0;i<n;i++)
    {
        wait[i]=0;
        prev[i]=arr[i][0];
       }
while(1)
      {
int ind=0,mini=100000;
for(int i=0;i<n;i++)
        {
            if(t>=arr[i][0]&&mini>arr[i][2]&&arr[i][1]>0)
         {
                  mini=arr[i][2];
                  ind=i;
         }
        }
if(mini==100000)
break;
arr[ind][1]-=1;
wait[ind]+=t-prev[ind];
        t++;
        prev[ind]=t;
      }
```

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

      {

         tat[i]=wait[i]+burst[i];

         averageTAT+=tat[i];

         averageWait+=wait[i];

      }

double a=averageTAT/(1.0*n), b=averageWait/(1.0*n);

printf("Process no.\t Arrival time \t Burst Time\t Priority_no \t Waiting Time \t
Turnaround Time\n");

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

{

  printf("%d\t\t %d\t\t %d \t\t %d
\t\t%d\t\t%d\n",i+1,arr[i][0],burst[i],arr[i][2],wait[i],tat[i]);
}

printf("The average wait time is: %lf and the average turn around time is:
%lf",b,a);

return 0;}
```

<u>NON PREMPTIVE</u>

```
#include <stdio.h>
void sort(int a[][3],int n)
{
for(int i=0;i<n;i++)
    {
        for(int j=i+1;j<n;j++)
        {
        if(a[i][2]<a[j][2])
        {
                int t=a[i][0];
                int k=a[i][1];
                int l=a[i][2];
                a[i][0]=a[j][0];
                a[i][1]=a[j][1];
                a[i][2]=a[j][2];
                a[j][0]=t;
                a[j][1]=k;
                a[j][2]=l;
        }
        }
```

```
     }
}
int main(){
int n;
printf("Enter the no. of process");
scanf("%d",&n);
int arr[n][3];
for(int i=0;i<n;i++)
     {
        printf("\n enter the arrival,burst time and priority number for the %dth
process\t",i+1);
         scanf("%d%d%d",&arr[i][0],&arr[i][1],&arr[i][2]);


     }
sort(arr,n);
int wait[n],tat[n],averageWait=0,averageTAT=0;
wait[0]=0;
for(int i=1;i<n;i++)
     {
        wait[i]=wait[i-1]+arr[i-1][1];
        averageWait+=wait[i];
     }
```

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

    {

       tat[i]=wait[i]+arr[i][1];

       averageTAT+=tat[i];

    }

double a=averageTAT/(1.0*n), b=averageWait/(1.0*n);

printf("Process no.\t Arrival time \t Burst Time\t Priority_no \t Waiting Time \t
Turn around Time\n");

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

{


printf(“%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n”,i+1,arr[i][0],arr[i][1],arr[i][2],wait[i],
tat[i]);

}

printf("The average wait time is: %lf and the average turn around time is:
%lf",b,a);

return 0;

}
```

11. Given a page reference string, write a program to find the page faults in this string by using FIFO page replacement policy. Where reference string and frame size will be entered by user.

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 5

int front = 0, back = -1, cs = 0, nf;

int f[MAX];

void enq(int x);

void deq(void);

void dis(void);

int isfound(int);

void main()

{

   int pf = 0, rfs, rf[15], i;

   printf("\n FIFO page replacement");

   printf("\n Enter the size of reference string:");

   scanf("%d", &rfs);

   printf("\n Enter the reference string:");

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

   {
       scanf("%d", &rf[i]);
   }   printf("\n Enter the number of free frames:");
   scanf("%d", &nf);
   enq(rf[0]);
   pf = 1;
   for (i = 0; i < rfs; i++) {
   if (!isfound(rf[i]))  {
        pf++;
        if (cs == nf)
                 deq();
        enq(rf[i]);
       }
   dis();
   }
printf("\n No of page faults :%d", pf);
}
int isfound(int x){
int i;
for (i = 0; i < cs; i++)
if (f[i] == x)
        return 1;
    return 0;
}
void enq(int x){
if (++back == nf)
back = 0;
f[back] = x;
cs++;
}
void dis(){
int i;
for (i = 0; i < cs; i++)
printf("%d", f[i]);
printf("\n");
}
void deq(){
    cs--;
    if (++front == nf)
    front = 0;
    return;
}
```

12. Given a page reference string, write a program to find the page faults in this string by using LRU page replacement policy. Where reference string and frame size will be entered by user.

```
#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

int fsize, ssize, f, frame[10], arrive[30], rstring[30];

int main()

{

int i, lfi, idx, cs = 0, f, ls = 0, pf = 0, j = 0, y, k, z = 0, time = 0;

int pagefound(int x);

void display();

int leastused();

int pagelocation(int x);

clrscr();

printf("\n\n\t\t LRU PAGE REPLACEMENT");

printf("\n\t\t --------------------");

printf("\n\n\t Enter the frame size:");

scanf("%d", &fsize);

printf("\n\t Enter the reference string size:");
```

```
scanf("%d", &ssize);

printf("\n\t Enter the reference string:");

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

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

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

   {

       frame[k] = -3;

       arrive[k] = 0;

   }

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

   {

     y = pagefound(rstring[i]);

       if (y == 0)

       {

        pf++;

        if (cs >= fsize)

          {

                 lfi = leastused();

                 frame[lfi] = rstring[i];

                 arrive[lfi] = ++time;

        }
```

```
         else if (cs < fsize)
         {
                frame[cs] = rstring[i];
                arrive[cs] = ++time;
         }
        }
        else
        {
         idx = pagelocation(rstring[i]);
         arrive[idx] = ++time;
        }
        cs++;
        display();
    }
    printf("\n Page fault=%d", pf);
}
int pagefound(int x)
{
    int i, val = 0;
    for (i = 0; i < fsize; i++)
    {
        if (x == frame[i])
        {
         val = 1;
         break;
        }
    }
    return (val);
}
void display(){
int i;
printf("\n");
for (i = 0; i < fsize; i++) {
        if (frame[i] >= 0)
        {
         printf("%d", frame[i]);
        }
        else
         printf("\t");
    }
}
int leastused(){
    int i, min = 0, n = 0;
    for (i = 1; i < fsize; i++)
    {
        if (arrive[i] < arrive[min]){
         min = i;
         n++;
        }  }
    if (n == 0)
        return (0);
    else
        return (min);
}
int pagelocation(int pageno){
    int i, flag = 0;
    for (i = 0; i < fsize; i++)  {
        if (frame[i] == pageno) {
         flag = 1;
         break;
        }
    }
    return (i);}
```

13. Given a page reference string, write a program to find the page faults in this string by using optimal page replacement policy. Where reference string and frame size will be entered by user.

```
#include <bits/stdc++.h>

using namespace std;

// Function to check whether a page exists

// in a frame or not

bool search(int key, vector<int>& fr)

{

        for (int i = 0; i < fr.size(); i++)

                if (fr[i] == key)

                        return true;

        return false;

}int predict(int pg[], vector<int>& fr, int pn, int index)

{

        // Store the index of pages which are going

        // to be used recently in future

        int res = -1, farthest = index;

        for (int i = 0; i < fr.size(); i++) {

                int j;

                for (j = index; j < pn; j++) {

                        if (fr[i] == pg[j]) {

                                if (j > farthest) {

                                        farthest = j;
                        res = i;
                    }
                    break;
                }
            }
            if (j == pn)
                return i;
        }
        return (res == -1) ? 0 : res;
}

void optimalPage(int pg[], int pn, int fn)
{       vector<int> fr;
        int hit = 0;
        for (int i = 0; i < pn; i++) {
            if (search(pg[i], fr)) {
                hit++;
                continue;
            }
            if (fr.size() < fn)
                fr.push_back(pg[i]);
            else {
                int j = predict(pg, fr, pn, i + 1);
```

```
                                fr[j] = pg[i];

                        }

                }

                cout << "No. of hits = " << hit << endl;

                cout << "No. of misses = " << pn - hit << endl;

}int main()

{

        int pg[20];

        int n;

        cout<<"Enter count of reference string \n";

        cin>>n;

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

   {

      cin>>pg[i];

   }

        int pn =n;

        int fn;

        cout<<"Enter number of frame \n";

        cin>>fn;

        optimalPage(pg, pn, fn);

        return 0;

}
```

14. Given an array of disk track numbers and initial head position, write a program to find the total number of seek operations done to access all the requested tracks if **First Come First Serve (FCFS)** disk scheduling algorithm is used.

```
#include <stdio.h>

int main(){

   int initial,n;

   printf("enter the initial position of the head and the no. of tracks to be
reached\n");

   scanf("%d%d",&initial,&n);


   printf("enter the disk track numbers to be reached\n");

   int arr[n];

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

   {

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

   }

   int total=initial-arr[0];

   if(total<0)

   total*=-1;

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

   {
        int a=arr[i]-arr[i+1];

        if(a<0)

        a*=-1;

         total+=a;

    }

    printf("The total amount is %d",total);


     return 0;

}
```

15. Given an array of disk track numbers and initial head position, write a program to find the total number of seek operations done to access all the requested tracks if **shortest seek time first** disk scheduling algorithm is used.

```
#include <stdio.h>

int main(){

  int initial,n;

   printf("enter the initial position of the head and the no. of tracks to be
reached\n");

   scanf("%d%d",&initial,&n);


   printf("enter the disk track numbers to be reached\n");

   int arr[n];

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

   {

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

   }

   int c=0,total=0;

   while(c!=n)

   {
        int d=10000,ind;
        for(int i=0;i<n;i++)
        {
         int l=initial-arr[i];
         if(l<0)
         l*=-1;

         if(l<d)
         {
         d=l;
         ind=i;
         }
        }
        initial=arr[ind];
        arr[ind]=10000;
        total+=d;
        c++;
    }
    printf("The total amount is %d",total);
     return 0;
}
```

16. Write a shell script to check whether the given number is even or odd.

```
echo "---- EVEN OR ODD IN SHELL SCRIPT -----"
echo -n "Enter a number:"
read n
echo -n "RESULT: "
if [ `expr $n % 2` == 0 ]
then
        echo "$n is even"
else
        echo "$n is Odd"
fi
```

24. WASS to check whether the given number is prime or not.

```
echo -e "Enter Number : \c"
read n
for((i=2; i<=$n/2; i++))
do
  ans=$(( n%i ))
  if [ $ans -eq 0 ]
  then
    echo "$n is not a prime number."
    exit 0
  fi
done
echo "$n is a prime number."
```

25. WASS to print reverse of a number and calculate sum of its digits.

```
echo enter n

read n

num=0

while [ $n -gt 0 ]

do

num=$(expr $num \* 10)

k=$(expr $n % 10)

num=$(expr $num + $k)

n=$(expr $n / 10)

done

echo” number is $num”

sum=0

while [ $num -gt 0 ]

do

    mod=$((num % 10))

    sum=$((sum + mod))

     num=$((num / 10))

done


echo “$sum”
```