FCFS

```
#include<stdio.h>
void main()
{
 int i,n,sum,wt,tat,twt,ttat;
 int t[10];
 float awt,atat;
 printf("Enter number of processors:\n");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
   printf("\n Enter the Burst Time of the process %d",i+1);
   scanf("\n %d",&t[i]);
 }
 printf("\n\n FIRST COME FIRST SERVE SCHEDULING ALGORITHM \n");
 printf("\n Process ID \t Waiting Time \t Turn Around Time \n");
 printf("1 \t\t 0 \t\t %d \n",t[0]);
 sum=0;
 twt=0;
 ttat=t[0];
 for(i=1;i<n;i++)
 {
  sum+=t[i-1];
  wt=sum;
  tat=sum+t[i];
  twt=twt+wt;
  ttat=ttat+tat;
  printf("\n %d \t\t %d \t\t %d",i+1,wt,tat);
  printf("\n\n");
 }
  awt=(float)twt/n;
  atat=(float)ttat/n;
  printf("\n Average Waiting Time %4.2f",awt);
  printf("\n Average Turnaround Time %4.2f",atat);
}
```

SJF (Non premptive)

```
#include<stdio.h>
void main()
{
int i,n,sum,wt,tat,twt,ttat;
 int t[10], p[10]; float awt,atat;
printf("Enter number of processors:\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\n Enter the Burst Time of the process %d",i+1);
scanf("\n %d",&t[i]);
}
for(i=0;i<n;i++)
p[i]=i;
for(i=0;i<n;i++)
{
for(int k=i+1;k<n;k++)
{
if(t[i]>t[k])
{
int temp;
temp=t[i];
t[i]=t[k];
t[k]=temp;
temp=p[i];
p[i]=p[k];
p[k]=temp;
}
}
}
printf("\n\n sjf SCHEDULING ALGORITHM \n");
printf("\n Process ID \t Waiting Time \t Turn Around Time \n");
printf("1 \t\t 0 \t\t %d \n",t[0]);
sum=0; twt=0; ttat=t[0];
for(i=1;i<n;i++)
{
sum+=t[i-1];
wt=sum;
tat=sum+t[i];
twt=twt+wt;
ttat=ttat+tat;
printf("\n %d \t\t %d \t\t %d",p[i],wt,tat);
printf("\n\n");
}
awt=(float)twt/n;
atat=(float)ttat/n;
printf("\n Average Waiting Time %4.2f",awt);
printf("\n Average Turnaround Time %4.2f",atat);
}
```

SJF (premptive)

```
#include <stdio.h>

void main()
{
   int t[10], at[10], wt[10], tat[10], n, index, remainingtime[10], bt;
   int currenttime = 0, sum = 0, twt = 0, ttat = 0;
   printf("Enter number of processes \n");
```

```
scanf("%d", &n);
for (int i = 0; i < n; i++)
{
    printf("Enter burst time of process %d: ", i);
    scanf("%d", &t[i]);
    printf("Enter arrival time of process %d: ", i);
    scanf("%d", &at[i]);
    remainingtime[i] = t[i];
}
printf("\n\nSJF SCHEDULING ALGORITHM\n");
printf("\nProcess ID \t Waiting Time \t Turn Around Time\n");
while (sum < n)
{
    index = -1;
    bt = 9999;
    for (int i = 0; i < n; i++)
    {
        if (remainingtime[i] < bt && currenttime >= at[i] && remainingtime[i] > 0)
        {
            index = i;
            bt = remainingtime[i];
        }
    }
    if (index != -1)
    {
        currenttime++;
        remainingtime[index]--;
        wt[index] = currenttime - at[index] - t[index];
        if (remainingtime[index] == 0)
        {
                sum++;

                tat[index] = currenttime - at[index];

            }

        }

        else

        {

            currenttime++;

        }

    }

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

    {

        printf("\n%d \t\t %d \t\t %d", i, wt[i], tat[i]);

        twt += wt[i];

        ttat += tat[i];

    }

    float awt, atat;

    awt = (float)twt / n;

    atat = (float)ttat / n;


    printf("\n\nAverage Waiting Time: %.2f", awt);

    printf("\nAverage Turnaround Time: %.2f", atat);

}
```

Round Robin

```
#include<stdio.h>

void main()
{
    int burst_time[10], arrival_time[10], remaining_time[10];
    int n, quantum, waiting_time[10] = {0}, turnaround_time[10] = {0};
    int total_waiting_time = 0, total_turnaround_time = 0;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    printf("Enter time quantum: ");
```

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

for(int i = 0; i < n; i++)
{
   printf("Enter burst time for process %d: ", i+1);
   scanf("%d", &burst_time[i]);

   printf("Enter arrival time for process %d: ", i+1);
   scanf("%d", &arrival_time[i]);

   remaining_time[i] = burst_time[i];
}

int current_time = 0;
int completed_processes = 0;

while(completed_processes < n)
{
   for(int i = 0; i < n; i++)
   {
      if(remaining_time[i] > 0)
      {
         if(remaining_time[i] <= quantum)
         {
            current_time += remaining_time[i];
            waiting_time[i] = current_time - burst_time[i] - arrival_time[i];
            turnaround_time[i] = current_time - arrival_time[i];
            remaining_time[i] = 0;
            completed_processes++;
         }
         else
         {
            current_time += quantum;
            remaining_time[i] -= quantum;
         }
      }
   }
}

for(int i = 0; i < n; i++)
{
   total_waiting_time += waiting_time[i];
   total_turnaround_time += turnaround_time[i];
}

float average_waiting_time = (float)total_waiting_time / n;
float average_turnaround_time = (float)total_turnaround_time / n;

printf("\nProcess\t\tWaiting Time\tTurnaround Time\n");
for(int i = 0; i < n; i++)
```

```
    {
        printf("P%d\t\t%d\t\t%d\n", i+1, waiting_time[i], turnaround_time[i]);
    }

    printf("\nAverage Waiting Time: %.2f\n", average_waiting_time);
    printf("Average Turnaround Time: %.2f\n", average_turnaround_time);
}
```

First Fit

```
#include<stdio.h>


void main()

{

   int frag[25], b[25], f[25], ff[25], bf[25] = {0};

   int i, j, nb, nf, temp;


   printf("\n\tMemory Management Scheme - First Fit\n");

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

   scanf("%d", &nb);

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

   scanf("%d", &nf);


   printf("\nEnter the size of the blocks:\n");

   for(i = 1; i <= nb; i++)

   {

      printf("Block %d: ", i);

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

   }


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

   for(i = 1; i <= nf; i++)

   {

      printf("File %d: ", i);

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

```
    }

    for(i = 1; i <= nf; i++)
    {
        for(j = 1; j <= nb; j++)
        {
            if(bf[j] != 1)
            {
                temp = b[j] - f[i];
                if(temp >= 0)
                {
                    ff[i] = j;
                    bf[j] = 1;
                    break;
                }
            }
        }
        frag[i] = temp;
    }

    printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment\n");
    for(i = 1; i <= nf; i++)
    {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i, f[i], ff[i], b[ff[i]], frag[i]);
    }
}
```

Best Fit

```
#include<stdio.h>
#include<conio.h>
void main()
{
```

```
int frag[25],b[25],f[25],i,j,nb,nf,temp,lowest=10000;

int bf[25]={0},ff[25];

printf("\nEnter the number of blocks:");

scanf("%d",&nb);

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

scanf("%d",&nf);

printf("\nEnter the size of the blocks:-\n");

for(i=1;i<=nb;i++)

{

printf("Block %d:",i);

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

}

printf("Enter the size of the files :-\n");

for(i=1;i<=nf;i++)

{

printf("File %d:",i);

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

}

for(i=1;i<=nf;i++)

{

for(j=1;j<=nb;j++)

{

if(bf[j]!=1)

{

temp=b[j]-f[i];

if(temp>=0)

{

if(lowest>temp)

{

ff[i]=j;

lowest=temp;
}
}
}}
frag[i]=lowest;
bf[ff[i]]=1;
lowest=10000;
}
printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment"); for(i=1;i<=nf && ff[i]!=0;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}
```

Worst Fit

```
#include<stdio.h>
#include<conio.h>

void main()
{
   int frag[25], b[25], f[25], ff[25];
   int i, j, nb, nf, temp, highest;
   static int bf[25] = {0};

   printf("\n\tMemory Management Scheme - Worst Fit\n");
   printf("Enter the number of blocks: ");
   scanf("%d", &nb);
   printf("Enter the number of files: ");
   scanf("%d", &nf);

   printf("\nEnter the size of the blocks:\n");
   for(i = 1; i <= nb; i++)
   {
       printf("Block %d: ", i);
       scanf("%d", &b[i]);
   }

   printf("Enter the size of the files:\n");
   for(i = 1; i <= nf; i++)
   {
       printf("File %d: ", i);
       scanf("%d", &f[i]);
   }

   for(i = 1; i <= nf; i++)
   {
       highest = -1; // Initialize highest to a negative value to ensure correct comparison
       for(j = 1; j <= nb; j++)
       {
           if(bf[j] != 1) // If block j is not allocated
           {
               temp = b[j] - f[i];
               if(temp >= 0 && temp > highest) // If file i can fit in block j and temp is higher than current
highest
               {
                   highest = temp;
                   ff[i] = j; // Store the index of the block with the highest remaining space
               }
           }
       }
       if(highest >= 0) // If a block is found
       {
           frag[i] = highest;
           bf[ff[i]] = 1; // Mark the block as allocated
       }
       else // If no block is found
       {
           frag[i] = 0;
       }
   }

   printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment\n");
   for(i = 1; i <= nf; i++)
   {
       printf("%d\t\t%d\t\t", i, f[i]);
       if(frag[i] > 0) // If the file is allocated
       {
           printf("%d\t\t%d\t\t%d\n", ff[i], b[ff[i]], frag[i]);
       }
       else // If the file is not allocated
       {
           printf("Not allocated\t0\t\t0\n");
       }
   }
}
```

FIFO

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
void main()
{

 printf("\n \t\t\t FIFI PAGE REPLACEMENT ALGORITHM");
 printf("\n Enter no.of frames....");
```

```
scanf("%d",&nof);
printf("Enter number of reference string..\n");
scanf("%d",&nor);
printf("\n Enter the reference string..");
for(i=0;i<nor;i++)
scanf("%d",&ref[i]);
printf("\nThe given reference string:");
for(i=0;i<nor;i++)
printf("%4d",ref[i]);
for(i=1;i<=nof;i++)
frm[i]=-1;
printf("\n");
for(i=0;i<nor;i++)
{
  flag=0;
  printf("\n\t Reference np%d->\t",ref[i]);
  for(j=0;j<nof;j++)
  {
    if(frm[j]==ref[i])
    {
      flag=1;
      break;
    }

  }
  if(flag==0)
  {
    pf++;
    victim++;
    victim=victim%nof;
    frm[victim]=ref[i];
    for(j=0;j<nof;j++)
    printf("%4d",frm[j]);
   }
 }
 printf("\n\n\t\t No.of pages faults...%d",pf);

}
```

LRU

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],lrucal[50],count=0;
int lruvictim();

void main()
{
  printf("\n\t\t\t LRU PAGE REPLACEMENT ALGORITHM");
  printf("\n Enter no.of Frames....");
  scanf("%d",&nof);

  printf(" Enter no.of reference string..");
  scanf("%d",&nor);

  printf("\n Enter reference string..");
  for(i=0;i<nor;i++)
  scanf("%d",&ref[i]);

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



  printf("\n\t The given reference string:");
printf("\n……………………………….");
for(i=0;i<nor;i++)
printf("%4d",ref[i]);
for(i=1;i<=nof;i++)
{
  frm[i]=-1;
  lrucal[i]=0;
}

for(i=0;i<10;i++)
{
 recent[i]=0;
 }
 printf("\n");
 for(i=0;i<nor;i++)
 {
   flag=0;
   printf("\n\t Reference NO %d->\t",ref[i]);
   for(j=0;j<nof;j++)
   {
      if(frm[j]==ref[i])
     {
        flag=1;
        break;
     }
   }

   if(flag==0)
   {
     count++;
     if(count<=nof)
```

```
       victim++;

       else

       victim=lruvictim();

       pf++;

       frm[victim]=ref[i];

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

       printf("%4d",frm[j]);

     }

     recent[ref[i]]=i;

   }

   printf("\n\n\t No.of page faults...%d",pf);

}

int lruvictim()

{

  int i,j,temp1,temp2;

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

  {

    temp1=frm[i];

    lrucal[i]=recent[temp1];

  }

  temp2=lrucal[0];

  for(j=1;j<nof;j++)

  {

    if(temp2>lrucal[j])

    temp2=lrucal[j];

  }

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

  if(ref[temp2]==frm[i])

  return i;

  return 0;

}
```

Bankers

```
#include<stdio.h>

#include<conio.h>


struct da

{

   int max[10], a1[10], need[10], before[10], after[10];

} p[10];


void main()

{

   int i, j, k, l, r, n, tot[10], av[10], cn = 0, cz = 0, temp = 0, c = 0;


   printf("\n ENTER THE NO. OF PROCESSES:");

   scanf("%d", &n);


   printf("\n ENTER THE NO. OF RESOURCES:");

   scanf("%d", &r);


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

   {

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

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

      {

         printf("MAXIMUM VALUE FOR RESOURCE %d:", j + 1);

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

      }


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

      {

         printf("ALLOCATED FROM RESOURCE %d:", j + 1);
```

```
        scanf("%d", &p[i].a1[j]);
        p[i].need[j] = p[i].max[j] - p[i].a1[j];
    }
}

for (i = 0; i < r; i++)
{
    printf("ENTER Available VALUE OF RESOURCE %d:", i + 1);
    scanf("%d", &av[i]);
}

printf("\nRESOURCES\tMAX\tALLOCATED\tNEEDED\tAVAIL");
for (i = 0; i < n; i++)
{
    printf("\n P%d \t", i + 1);
    printf("\t");
    for (j = 0; j < r; j++)
        printf("%d", p[i].max[j]);
    printf("\t");
    for (j = 0; j < r; j++)
        printf("%d", p[i].a1[j]);
    printf("\t");
    for (j = 0; j < r; j++)
        printf("%d", p[i].need[j]);
    printf("\t");
    for (j = 0; j < r; j++)
    {
        if (i == 0)
            printf("%d", av[j]);
    }
}
```

```
printf("\n\nProcess\tAVAIL BEFORE\tAVAIL AFTER ");
for (l = 0; l < n; l++)
{
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < r; j++)
        {
            if (p[i].need[j] > av[j])
                cn++;
            if (p[i].max[j] == 0)
                cz++;
        }

        if (cn == 0 && cz != r)
        {
            for (j = 0; j < r; j++)
            {
                p[i].before[j] = av[j];
                p[i].after[j] = p[i].before[j] + p[i].a1[j];
                av[j] = p[i].after[j];
                p[i].max[j] = 0;
            }

            printf("\nP %d \t", i + 1);
            for (j = 0; j < r; j++)
                printf("%d", p[i].before[j]);
            printf("\t");
            for (j = 0; j < r; j++)
                printf("%d", p[i].after[j]);
            cn = 0;
```

```
                cz = 0;
                c++;
                break;
            }
            else
            {
                cn = 0;
                cz = 0;
            }
        }
    }

    if (c == n)
        printf("\n THE ABOVE SEQUENCE IS A SAFE SEQUENCE");
    else
        printf("\n DEADLOCK OCCURRED");
}
```

PRODUCER CONSUMER

```
#include<stdio.h>
void main()
{
int buffer[10], bufsize, in, out, produce, consume,
choice=0; in = 0;
out = 0;
bufsize = 10;
while(choice !=4)
{
printf("\n1. Produce \t 2. Consume \t3. Print\t 4. Exit");
printf("\nEnter your choice: ");
scanf("%d",&choice);
```

```
switch(choice) {
case 1: if((in+1)%bufsize==out)
printf("\nBuffer is Full");
else
{
printf("\nEnter the value: ");
scanf("%d", &produce);
buffer[in] = produce;
in = (in+1)%bufsize;
}
break;
case 2: if(in == out)
printf("\nBuffer is Empty");
else
{
consume = buffer[out];
printf("\nThe consumed value is %d", consume);
out = (out+1)%bufsize;
}
break;
case 3: printf("\nBuffer Contents: ");
        if(in == out)
           printf("\nBuffer is Empty");
        else {
           int i = out;
           while(i != in) {
              printf("%d ", buffer[i]);
              i = (i + 1) % bufsize;
           }
        }
        break;
}

}

}
```