LAB PROGRAMS

*CPU SCHECULING ALGORITHMS

1.FCFS ALGORITHM (without arrival times)

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
int main()
{
   int n;
   cout<<"Enter the number of process: ";
   cin>>n;
   int a[n];
   for(int i=0;i<n;i++)
   {
      cout<<"Enter the burst time of process "<<i+1<<": ";
      cin>>a[i];
   }
   cout<<endl;
   cout<<"FIRST COME FIRST SERVE CPU SCHEDULING ALGORITHM:"<<endl;
   cout<<"Process ID     Waiting Time      Turnaround Time"<<endl;
   int wait_time=0,turn_time=0,avg_wt=0,avg_tt=0;
   for(int i=0;i<n;i++)
   {
      turn_time+=a[i];
      avg_tt+=turn_time;
      avg_wt+=wait_time;
      cout<<i+1<<"               "<<wait_time<<"               "<<turn_time<<endl;
      wait_time+=a[i];
   }
   cout<<endl;
   cout<<"Average waiting time= "<<(float)avg_wt/n<<endl;
   cout<<"Average turnaround time= "<<(float)avg_tt/n<<endl;
   return 0;
}
```

```
Enter the number of process: 4
Enter the burst time of process 1: 16
Enter the burst time of process 2: 12
Enter the burst time of process 3: 24
Enter the burst time of process 4: 8

FIRST COME FIRST SERVE CPU SCHEDULING ALGORITHM:
Process ID     Waiting Time      Turnaround Time
1                 0                 16
2                 16                 28
3                 28                 52
4                 52                 60

Average waiting time= 24
Average turnaround time= 39
```

2. FCFS ALGORITHM (WITH ARRIVAL TIMES)

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
int main()
{
   int n;
   cout<<"Enter the number of process: ";
   cin>>n;
   int at[n];
   cout<<"Enter the arrival time of the processes: "<<endl;
   for(int i=0;i<n;i++)
   {
      cout<<"Enter the arrival time of process "<<i+1<<": ";
      cin>>at[i];
   }
   int bt[n];
   for(int i=0;i<n;i++)
   {
      cout<<"Enter the burst time of process "<<i+1<<": ";
      cin>>bt[i];
   }
   map<int,map<int,int>>m;
   map<int,map<int,int>>::iterator it;
```

```
   map<int,int>::iterator itr;
   for(int i=0;i<n;i++)
   {
      m.insert(make_pair(at[i],map<int,int>()));
      m[at[i]].insert(make_pair(i,bt[i]));
   }

   cout<<endl;
   cout<<"FIRST COME FIRST SERVE CPU SCHEDULING ALGORITHM:"<<endl;
   cout<<"Process ID\tArrival Time\tBurst Time\tCompletion Time \tWaiting
Time\tTurnaround Time"<<endl;
   int wt[n]={0},tt[n]={0},ct[n]={0},total_wt=0,total_tt=0;
   int i=1;
   for (it=m.begin();it!=m.end();it++)
   {
      itr=it->second.begin();
      if(it->first<ct[i-1])
      {
         ct[i]=ct[i-1]+itr->second;
      }
      else
      {
         ct[i]=it->first+itr->second;
      }
      tt[i]=ct[i]-it->first;
      wt[i]=tt[i]-itr->second;
      total_wt+=wt[i];
      total_tt+=tt[i];
      cout<<itr->first<<"\t\t"<<it->first<<"\t\t"
      <<itr->second<<"\t\t"<<ct[i]<<
      "\t\t"<<wt[i]<<"\t\t"<<tt[i]<<endl;
      i++;
   }
   cout<<endl;
   cout<<"Average waiting time= "<<(float)total_wt/n<<endl;
   cout<<"Average turnaround time= "<<(float)total_tt/n<<endl;
```

```
    return 0;
}
```

```
Enter the number of process: 4
Enter the arrival time of the processes:
Enter the arrival time of process 1: 0
Enter the arrival time of process 2: 2
Enter the arrival time of process 3: 4
Enter the arrival time of process 4: 6
Enter the burst time of process 1: 15
Enter the burst time of process 2: 12
Enter the burst time of process 3: 9
Enter the burst time of process 4: 6

FIRST COME FIRST SERVE CPU SCHEDULING ALGORITHM:
Process ID      Arrival Time    Burst Time      Completion Time Waiting Time    Turnaround Time
0               0               15              15                      0               15
1               2               12              27                      13              25
2               4               9               36                      23              32
3               6               6               42                      30              36

Average waiting time= 16.5
Average turnaround time= 27
```

3.SJF ALGORITHM (WITHOUT ARRIVAL TIMES)

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
int main()
{
   int n;
   cout<<"Enter the number of process: ";
   cin>>n;
   map<int,int>m;
   for(int i=1;i<=n;i++)
   {
      cout<<"Enter the burst time of process "<<i<<": ";
      cin>>m[i];
   }
   multimap<int,int>mm;
```

```
   map<int,int>::iterator it;
   for (it=m.begin();it!=m.end();it++)
{
mm.insert(make_pair(it->second,it->first));
}
   cout<<endl;
   cout<<"SHORTEST JOB FIRST CPU SCHEDULING ALGORITHM:"<<endl;
   cout<<"Process ID     Waiting Time      Turnaround Time"<<endl;
   int wait_time=0,turn_time=0,avg_wt=0,avg_tt=0;
   multimap<int,int>::iterator i;
   for(i=mm.begin();i!=mm.end();i++)
   {
      turn_time+=i->first;
      avg_tt+=turn_time;
      avg_wt+=wait_time;
      cout<<i->second<<"               "<<wait_time<<"
"<<turn_time<<endl;
      wait_time+=i->first;
   }
   cout<<endl;
   cout<<"Average waiting time= "<<(float)avg_wt/n<<endl;
   cout<<"Average turnaround time= "<<(float)avg_tt/n<<endl;
   return 0;
}
```

```
Enter the number of process: 4
Enter the burst time of process 1: 8
Enter the burst time of process 2: 12
Enter the burst time of process 3: 6
Enter the burst time of process 4: 24

SHORTEST JOB FIRST CPU SCHEDULING ALGORITHM:
Process ID     Waiting Time      Turnaround Time
3                  0                  6
1                  6                  14
2                  14                  26
4                  26                  50

Average waiting time= 11.5
Average turnaround time= 24
```

4.SJF ALGORITHM (WITH ARRIVAL TIMES)

```
#include <bits/stdc++.h>
using namespace std;
struct Process
{
	int pid;
	int bt;
	int art;
};

void findWaitingTime(Process proc[], int n,int wt[])
{
	int rt[n];
	for (int i = 0; i < n; i++)
	   rt[i] = proc[i].bt;

	int complete = 0, t = 0, minm = INT_MAX;
	int shortest = 0, finish_time;
	bool check = false;
	while (complete != n)
	{
```

```
                for (int j = 0; j < n; j++) {
                        if ((proc[j].art <= t) &&
                        (rt[j] < minm) && rt[j] > 0) {
                                minm = rt[j];
                                shortest = j;
                                check = true;
                        }
                }

                if (check == false) {
                        t++;
                        continue;
                }

                rt[shortest]--;
                minm = rt[shortest];
                if (minm == 0)
                        minm = INT_MAX;

                if (rt[shortest] == 0)
                {
                        complete++;
                        check = false;
                        finish_time = t + 1;
                        wt[shortest] = finish_time -proc[shortest].bt -
proc[shortest].art;

                        if (wt[shortest] < 0)
                                wt[shortest] = 0;
                }
                t++;
        }
}

void findTurnAroundTime(Process proc[], int n,int wt[], int tat[])
{
        for (int i = 0; i < n; i++)
                tat[i] = proc[i].bt + wt[i];
}
```

```
void findavgTime(Process proc[], int n)
{
	int wt[n], tat[n], total_wt = 0,total_tat = 0;
	findWaitingTime(proc, n, wt);
	findTurnAroundTime(proc, n, wt, tat);
	cout << " P\t\t"<< "BT\t\t"<< "WT\t\t"<< "TAT\t\t\n";
	for (int i = 0; i < n; i++) {
		total_wt = total_wt + wt[i];
		total_tat = total_tat + tat[i];
		cout << " " << proc[i].pid << "\t\t"
			<< proc[i].bt << "\t\t " << wt[i]
			<< "\t\t " << tat[i] << endl;
	}

	cout << "\nAverage waiting time = "<< (float)total_wt / (float)n;
	cout << "\nAverage turn around time = "<< (float)total_tat / (float)n;
}


int main()
{
	Process proc[] = { { 1, 6, 2 }, { 2, 2, 5 },{ 3, 8, 1 }, { 4, 3, 0}, {5, 4, 4} };
	int n = sizeof(proc) / sizeof(proc[0]);
	findavgTime(proc, n);
	return 0;
}
```

```
 P		BT		WT		TAT
 1		6		 7		 13
 2		2		 0		 2
 3		8		 14		 22
 4		3		 0		 3
 5		4		 2		 6

Average waiting time = 4.6
Average turn around time = 9.2
```

5.ROUND ROBIN ALGORITHM (WITHOUT ARRIVAL TIMES)

```
#include <bits/stdc++.h>
using namespace std;
```

```
void findWaitingTime(int processes[], int n,int bt[], int wt[], int quantum)
{
   int rem_bt[n];
   for (int i = 0 ; i < n ; i++)
      rem_bt[i] = bt[i];
   int t = 0;
   while (1)
   {
      bool done = true;
      for (int i = 0 ; i < n; i++)
      {
         if (rem_bt[i] > 0)
         {
            done = false;
            if (rem_bt[i] > quantum)
            {
               t += quantum;
               rem_bt[i] -= quantum;
            }
            else
            {
               t = t + rem_bt[i];
               wt[i] = t - bt[i];
               rem_bt[i] = 0;
            }
         }
      }
      if (done == true)
      break;
   }
}

void findTurnAroundTime(int processes[], int n,int bt[], int wt[], int tat[])
{
   for (int i = 0; i < n ; i++)
      tat[i] = bt[i] + wt[i];
}

void findavgTime(int processes[], int n, int bt[],int quantum)
{
```

```
    int wt[n], tat[n], total_wt = 0, total_tat = 0;
    findWaitingTime(processes, n, bt, wt, quantum);
    findTurnAroundTime(processes, n, bt, wt, tat);
    cout << "PN\t "<< " \tBT "<< " WT " << " \tTAT\n";
    for (int i=0; i<n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << i+1 << "\t\t" << bt[i] <<"\t "<< wt[i] <<"\t\t " << tat[i] <<endl;
    }
    cout << "Average waiting time = "<< (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "<< (float)total_tat / (float)n;
}

int main()
{
    int n;
    cout<<"Enter the number of processes: ";
    cin>>n;
    int processes[n],burst_time[n];
    for(int i=0;i<n;i++)
    {
        processes[i]=i+1;
    }
    for(int i=0;i<n;i++)
    {
        cout<<"Enter the burst time of process "<<i+1<<": ";
        cin>>burst_time[i];
    }
    int quantum;
    cout<<"Enter the time quantum: ";
    cin>>quantum;
    findavgTime(processes, n, burst_time, quantum);
}
```

```
Enter the number of processes: 4
Enter the burst time of process 1: 15
Enter the burst time of process 2: 3
Enter the burst time of process 3: 48
Enter the burst time of process 4: 12
Enter the time quantum: 3
PN              BT  WT          TAT
 1              15      27              42
 2              3       3               6
 3              48      30              78
 4              12      27              39
Average waiting time = 21.75
Average turn around time = 41.25
```

6. ROUND ROBIN (WITH ARRIVAL TIMES)

```
#include <iostream>
using namespace std;

void queueUpdation(int queue[],int timer,int arrival[],int n, int
maxProccessIndex){
        int zeroIndex;
        for(int i = 0; i < n; i++){
                if(queue[i] == 0){
                        zeroIndex = i;
                        break;
                }
        }
        queue[zeroIndex] = maxProccessIndex + 1;
}

void queueMaintainence(int queue[], int n){
        for(int i = 0; (i < n-1) && (queue[i+1] != 0) ; i++){
                int temp = queue[i];
                queue[i] = queue[i+1];
                queue[i+1] = temp;
        }
}

void checkNewArrival(int timer, int arrival[], int n, int maxProccessIndex,int
queue[]){
```

```
        if(timer <= arrival[n-1]){
        bool newArrival = false;
        for(int j = (maxProccessIndex+1); j < n; j++){
                        if(arrival[j] <= timer){
                        if(maxProccessIndex < j){
                                maxProccessIndex = j;
                                newArrival = true;
                        }
                }
        }
        if(newArrival)
                queueUpdation(queue,timer,arrival,n, maxProccessIndex);
        }
}
int main(){
        int n,tq, timer = 0, maxProccessIndex = 0;
        float avgWait = 0, avgTT = 0;
        cout << "\nEnter the time quantum : ";
        cin>>tq;
        cout << "\nEnter the number of processes : ";
        cin>>n;
        int arrival[n], burst[n], wait[n], turn[n], queue[n], temp_burst[n];
        bool complete[n];

        cout << "\nEnter the arrival time of the processes : ";
        for(int i = 0; i < n; i++)
                cin>>arrival[i];

        cout << "\nEnter the burst time of the processes : ";
        for(int i = 0; i < n; i++){
                cin>>burst[i];
                temp_burst[i] = burst[i];
        }

        for(int i = 0; i < n; i++){
                complete[i] = false;
                queue[i] = 0;
        }
        while(timer < arrival[0])
                timer++;
```

```
        queue[0] = 1;

        while(true){
                bool flag = true;
                for(int i = 0; i < n; i++){
                        if(temp_burst[i] != 0){
                                flag = false;
                                break;
                        }
                }
                if(flag)
                        break;

                for(int i = 0; (i < n) && (queue[i] != 0); i++){
                        int ctr = 0;
                        while((ctr < tq) && (temp_burst[queue[0]-1] > 0)){
                                temp_burst[queue[0]-1] -= 1;
                                timer += 1;
                                ctr++;
                                checkNewArrival(timer, arrival, n, maxProccessIndex,
queue);
                        }
                        if((temp_burst[queue[0]-1] == 0) && (complete[queue[0]-1]
== false)){
                                turn[queue[0]-1] = timer;
                                complete[queue[0]-1] = true;
                        }
                        bool idle = true;
                        if(queue[n-1] == 0){
                                for(int i = 0; i < n && queue[i] != 0; i++){
                                        if(complete[queue[i]-1] == false){
                                                idle = false;
                                        }
                                }
                        }
                        else
                                idle = false;

                        if(idle){
                                timer++;
```

```
                        checkNewArrival(timer, arrival, n, maxProccessIndex,
queue);
                }
                queueMaintainence(queue,n);
            }
        }

        for(int i = 0; i < n; i++){
            turn[i] = turn[i] - arrival[i];
            wait[i] = turn[i] - burst[i];
        }

        cout << "\nProgram No.\tArrival Time\tBurst Time\tWait
Time\tTurnAround Time"
            << endl;
        for(int i = 0; i < n; i++){
            cout<<i+1<<"\t\t"<<arrival[i]<<"\t\t"
            <<burst[i]<<"\t\t"<<wait[i]<<"\t\t"<<turn[i]<<endl;
        }
        for(int i =0; i< n; i++){
            avgWait += wait[i];
            avgTT += turn[i];
        }
        cout<<"\nAverage wait time : "<<(avgWait/n)<<"\nAverage Turn Around
Time : "<<(avgTT/n);

        return 0;

}
```

```
Enter the time quantum : 2

Enter the number of processes : 4

Enter the arrival time of the processes : 0 1 2 3

Enter the burst time of the processes : 5 4 2 1

Program No.     Arrival Time    Burst Time      Wait Time       TurnAround Time
1               0               5               7               12
2               1               4               6               10
3               2               2               2               4
4               3               1               5               6

Average wait time : 5
Average Turn Around Time : 8
```

7.PRIORITY ALGORITHM (WITHOUT ARRIVAL TIMES)

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
   int n;
   cout<<"Enter number of process: ";
   cin>>n;
   int p[n],bt[n];
   for(int i=0;i<n;i++)
   {
      cout<<"Enter the priority of process "<<i+1<<": ";
      cin>>p[i];
      cout<<"Enter the burst time of process "<<i+1<<": ";
      cin>>bt[i];
   }
   vector<pair<int,int>> v;
   for(int i=0;i<n;i++)
   {
      v.push_back({p[i],bt[i]});
   }
   sort(v.begin(),v.end());
   int tat[n],wt[n],twt=0,ttat=0;
   tat[0]=v[0].second;
   wt[0]=0;
   for(int i=1;i<n;i++)
   {
      tat[i]=tat[i-1]+v[i].second;
      wt[i]=tat[i]-v[i].second;
   }
   cout<<"PID\tPRIORITY\tBT\tTAT\tWT"<<endl;
   for(int i=0;i<n;i++)
   {
      cout<<i+1<<"\t"<<p[i]<<"\t\t"<<bt[i]<<"\t";
      for(int j=0;j<n;j++)
      {
         if(v[j].first==p[i])
         {
            cout<<tat[j]<<"\t"<<wt[j]<<endl;
         }
```

```
    }
  }
  for(int i=0;i<n;i++)
  {
    twt+=wt[i];
    ttat+=tat[i];
  }
  cout <<"Average waiting time = "<< (float)twt/n;
  cout <<"\nAverage turn around time = "<< (float)ttat/n;
  return 0;
}
```

```
Enter number of process: 4
Enter the priority of process 1: 3
Enter the burst time of process 1: 12
Enter the priority of process 2: 4
Enter the burst time of process 2: 16
Enter the priority of process 3: 1
Enter the burst time of process 3: 13
Enter the priority of process 4: 2
Enter the burst time of process 4: 9
PID     PRIORITY        BT      TAT     WT
1       3               12      34      22
2       4               16      50      34
3       1               13      13      0
4       2               9       22      13
Average waiting time = 17.25
Average turn around time = 29.75
```

8.PRIORITY ALGORITHM (WITH ARRIVAL TIMES)

```
#include <bits/stdc++.h>
using namespace std;
#define totalprocess 5

struct process
{
int at,bt,pr,pno;
};

process proc[50];
```

```
bool comp(process a,process b)
{
if(a.at == b.at)
{
return a.pr<b.pr;
}
else
{
        return a.at<b.at;
}
}
void get_wt_time(int wt[])
{
int service[50];
service[0] = proc[0].at;
wt[0]=0;

for(int i=1;i<totalprocess;i++)
{
service[i]=proc[i-1].bt+service[i-1];

wt[i]=service[i]-proc[i].at;
        if(wt[i]<0)
        {
        wt[i]=0;
        }
}

}

void get_tat_time(int tat[],int wt[])
{
for(int i=0;i<totalprocess;i++)
{
        tat[i]=proc[i].bt+wt[i];
}

}

void findgc()
```

```
{
int wt[50],tat[50];
double wavg=0,tavg=0;
get_wt_time(wt);
get_tat_time(tat,wt);
int stime[50],ctime[50];

stime[0] = proc[0].at;
ctime[0]=stime[0]+tat[0];

for(int i=1;i<totalprocess;i++)
        {
                stime[i]=ctime[i-1];
                ctime[i]=stime[i]+tat[i]-wt[i];
        }

cout<<"Process_no\tStart_time\tComplete_time\tTurn_Around_Time\tWaitin
g_Time"<<endl;
for(int i=0;i<totalprocess;i++)
        {
                wavg += wt[i];
                tavg += tat[i];

                cout<<proc[i].pno<<"\t\t"<<
                        stime[i]<<"\t\t"<<ctime[i]<<"\t\t"<<
                        tat[i]<<"\t\t\t"<<wt[i]<<endl;
        }
        cout<<"Average waiting time is : ";
        cout<<wavg/(float)totalprocess<<endl;
        cout<<"average turnaround time : ";
        cout<<tavg/(float)totalprocess<<endl;

}

int main()
{
int arrivaltime[] = { 1, 2, 3, 4, 5 };
int bursttime[] = { 3, 5, 1, 7, 4 };
int priority[] = { 3, 4, 1, 7, 8 };
```

```
for(int i=0;i<totalprocess;i++)
{
        proc[i].at=arrivaltime[i];
        proc[i].bt=bursttime[i];
        proc[i].pr=priority[i];
        proc[i].pno=i+1;
}
        sort(proc,proc+totalprocess,comp);
        findgc();
        return 0;
}
```

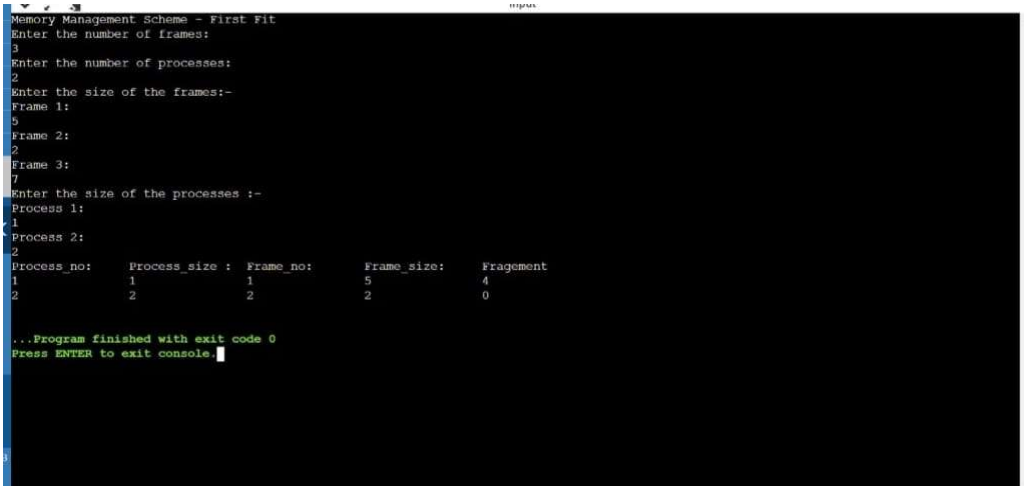
```
Process_no      Start_time      Complete_time   Turn_Around_Time        Waiting_Time
1               1               4               3                       0
2               4               9               7                       2
3               9               10              7                       6
4               10              17              13                      6
5               17              21              16                      12
Average waiting time is : 5.2
average turnaround time : 9.2
```

*FIT ALGORITHMS

9.FIRST FIT ALGORITHM

```
#include<iostream>
#define max 25
using namespace std;
int main()
{
   int frag[max],b[max],f[max],i,j,nb,nf,temp;
   static int bf[max],ff[max];
   cout<<"Memory Management Scheme - First Fit"<<endl;
   cout<<"Enter the number of frames:"<<endl;
   cin>>nb;
   cout<<"Enter the number of processes:"<<endl;
   cin>>nf;
   cout<<"Enter the size of the frames:-"<<endl;
   for(i=1;i<=nb;i++)
   {
      cout<<"Frame "<<i<<":"<<endl;
      cin>>b[i];
   }
```

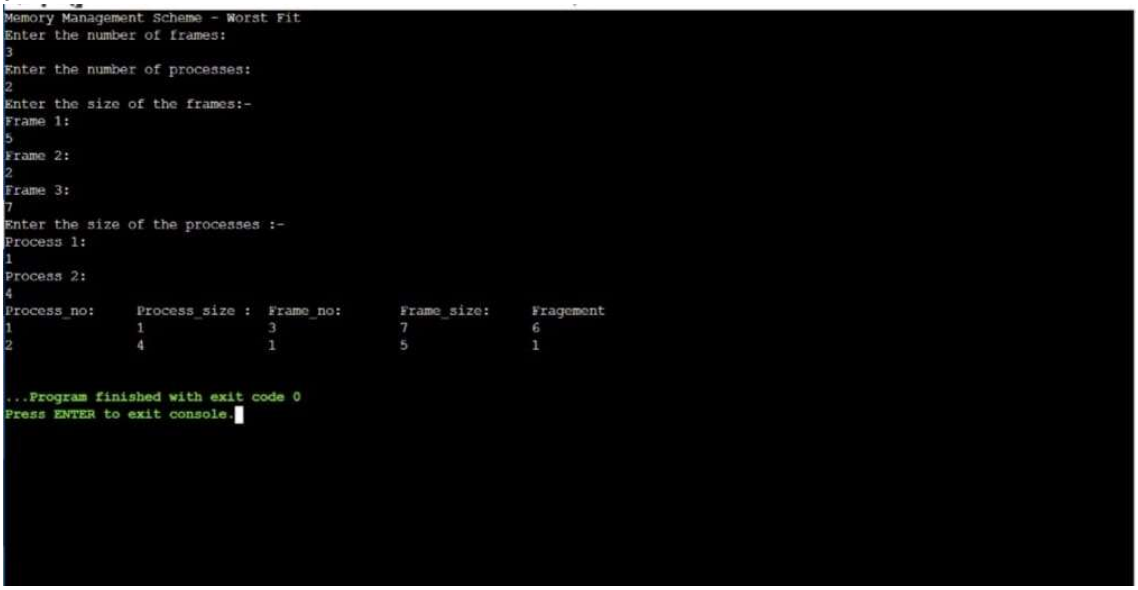
```
   cout<<"Enter the size of the processes :-"<<endl;
   for(i=1;i<=nf;i++)
   {
      cout<<"Process "<<i<<":"<<endl;
      cin>>f[i];
   }
   for(i=1;i<=nf;i++)
   {
      for(j=1;j<=nb;j++)
      {
         if(bf[j]!=1)
         {
            temp=b[j]-f[i];
            if(temp>=0)
            {
               ff[i]=j;
               break;
            }
         }
      }
      frag[i]=temp;
      bf[ff[i]]=1;
   }
   cout<<"Process_no:\tProcess_size
:\tFrame_no:\tFrame_size:\tFragement"<<endl;
   for(i=1;i<=nf;i++)
   cout<<i<<"\t\t"<<f[i]<<"\t\t"<<ff[i]<<"\t\t"<<b[ff[i]]<<"\t\t"<<frag[i]<<endl;
   return 0;
}
```



10.WORST FIT ALGORITHM

```
#include<iostream>
#define max 25
using namespace std;
int main()
{
   int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
   static int bf[max],ff[max];
   cout<<"Memory Management Scheme - Worst Fit"<<endl;
   cout<<"Enter the number of frames:"<<endl;
   cin>>nb;
   cout<<"Enter the number of processes:"<<endl;
   cin>>nf;
   cout<<"Enter the size of the frames:-"<<endl;
   for(i=1;i<=nb;i++)
   {
      cout<<"Frame "<<i<<":"<<endl;
      cin>>b[i];
   }
   cout<<"Enter the size of the processes :-"<<endl;
   for(i=1;i<=nf;i++)
   {
      cout<<"Process "<<i<<":"<<endl;
      cin>>f[i];
   }
   for(i=1;i<=nf;i++)
   {
      for(j=1;j<=nb;j++)
      {
         if(bf[j]!=1) //if bf[j] is not allocated
         {
            temp=b[j]-f[i];
            if(temp>=0)
               if(highest<temp)
               {
                  ff[i]=j;
                  highest=temp;
               }
         }
      }
```

```
      frag[i]=highest;
      bf[ff[i]]=1;
      highest=0;
   }
   cout<<"Process_no:\tProcess_size
:\tFrame_no:\tFrame_size:\tFragement"<<endl;
   for(i=1;i<=nf;i++)
   cout<<i<<"\t\t"<<f[i]<<"\t\t"<<ff[i]<<"\t\t"<<b[ff[i]]<<"\t\t"<<frag[i]<<endl;
   return 0;
}
```



11.BEST FIT ALGORITHM

```
#include<iostream>
#define max 25
using namespace std;
int main()
{
   int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
   static int bf[max],ff[max];
   cout<<"Memory Management Scheme - Best Fit"<<endl;
   cout<<"Enter the number of frames:"<<endl;
   cin>>nb;
   cout<<"Enter the number of processes:"<<endl;
   cin>>nf;
   cout<<"Enter the size of the frames:-"<<endl;
   for(i=1;i<=nb;i++)
   {
```

```
    cout<<"Frame "<<i<<":"<<endl;
    cin>>b[i];
  }
  cout<<"Enter the size of the processes :-"<<endl;
  for(i=1;i<=nf;i++)
  {
    cout<<"Process "<<i<<":"<<endl;
    cin>>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;
  }
  cout<<"Process_no:\tProcess_size
:\tFrame_no:\tFrame_size:\tFragement"<<endl;
  for(i=1;i<=nf;i++)
  cout<<i<<"\t\t"<<f[i]<<"\t\t"<<ff[i]<<"\t\t"<<b[ff[i]]<<"\t\t"<<frag[i]<<endl;
  return 0;
}
```

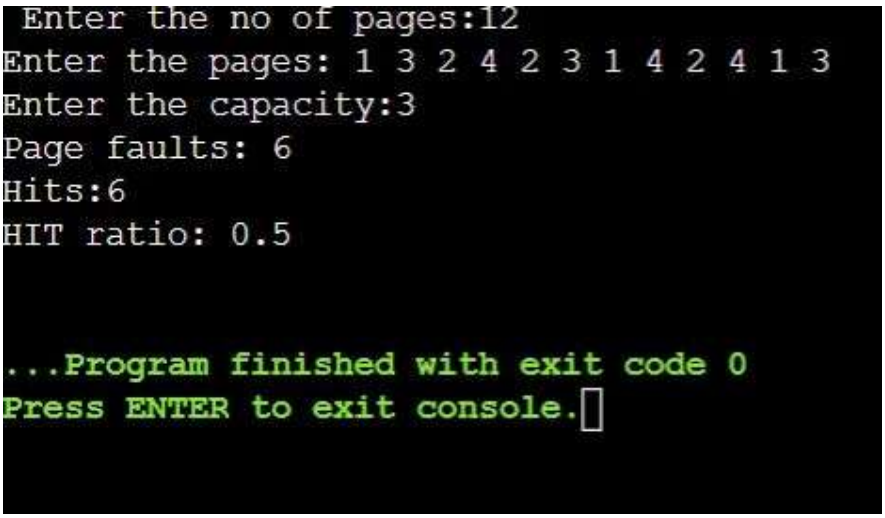
*PAGE REPLACEMENT ALGORITHMS

12. FIRST IN FIRST OUT ALGORITHM

```
#include <bits/stdc++.h>
using namespace std;
int pagefaults (int pages[ ], int n, int capacity)
{
unordered_set<int>s;
queue <int> indexes;
int page_faults=0 ;
for (int i=0; i<n; i++)
{
   if (s.size () < capacity)
   {
      if (s.find(pages [i])== s.end()){
         s.insert (pages[i]);
         page_faults++;
         indexes.push(pages[i]);
      }
   }
   else
   {
      if(s.find(pages[i])== s.end())
      {
         int val=indexes.front();
         indexes.pop();
```

```
        s.erase (val);
        s.insert (pages[i]);

        indexes.push (pages [i]);
        page_faults++;
      }
    }
}
return page_faults;
}
int main()
{
int page;
cout <<" Enter the no of pages:";
cin>> page; int pages [page];
cout<<"Enter the pages: ";
for (int i=0; i<page; i++)
cin>>pages[i];
int capacity;
cout << "Enter the capacity:";
cin>> capacity;
int f=pagefaults(pages, page, capacity);
cout << "Page faults: "<<f<<endl;
cout<<"Hits:" << page -f<<endl;
cout<<"HIT ratio: "<< float(page-f) / float(page)<<endl;
return 0;
}
```



13. LEAST RECENTLY USED ALGORITHM

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
int main()
{
int page;
cout<<"Enter the no of pages:";
cin>> page;
int arr[page];
cout << "Enter the pages:";
for (int i=0; i<page; i++)
cin>> arr[i];
int capacity,page_faults=0;
cout << "Enter the capacity:";

cin>> capacity;
deque <int>q;
deque <int>:: iterator itr;
q.clear();
for (int i: arr)
{
  itr = find (q.begin (), q.end (), i);
  if(!(itr != q.end()))
  {
     ++page_faults;
     if (q.size() == capacity)
     {
       q.erase (q.begin());
       q.push_back(i);
     }
    else
   {
      q.push_back (i);
   }
  }
  else
  {
     q.erase (itr);
     q.push_back(i);
```

```
  }
}
cout << "No. of page faults "<< page_faults <<endl;
cout<< "No. of hits :"<<page-page_faults <<endl;
cout<<" HIT ratio: "<< float (page-page_faults)/float (page)<<endl;
return 0;
}
```

```
Enter the no of pages:12
Enter the pages:1 3 2 4 2 3 1 4 2 4 1 3
Enter the capacity:3
No. of page faults 8
No. of hits :4
 HIT ratio: 0.333333


...Program finished with exit code 0
Press ENTER to exit console.
```

14.OPTIMAL ALGORITHM

```
#include <bits/stdc++.h>
using namespace std;

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)
{
   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;
cout<<"No of HIT ratio = "<<float(hit)/ float (pn) <<endl;
}
```

```
int main()
{
int page;
cout<<"Enter the no of pages:";
cin>>page;
int pg [page];
cout<<"Enter the pages:";
for(int i=0;i<page;i++)
cin>>pg[i];
int capacity;
cout<<"enter the capacity:";
cin>>capacity;
optimalpage (pg, page, capacity);
return 0;
}
```

```
Enter the no of pages:12
Enter the pages:1 3 2 4 2 3 1 4 2 4 1 3
Enter the capacity:3
No. of page faults 8
No. of hits :4
 HIT ratio: 0.333333


...Program finished with exit code 0
Press ENTER to exit console.
```

*DISC SCHEDULING ALGORITHMS

15.SCAN ALGORITHM

```
#include <bits/stdc++.h>
using namespace std;

void scan(int a[],int n,int head,string dir)
{
   vector<int>left,right,track;
   int dis=0,curr;
   if(dir=="left")
      left.push_back(0);
```

```
else if(dir=="right")
   right.push_back(a[n-1]+1);
for(int i=0;i<n;i++)
{
   if(a[i]>head)
      right.push_back(a[i]);
   else if(a[i]<head)
      left.push_back(a[i]);
}
sort(left.begin(),left.end());
sort(right.begin(),right.end());
int t=2;
while(t--)
{
   if(dir=="left")
   {
      for(int i=left.size()-1;i>=0;i--)
      {
         curr=left[i];
         track.push_back(curr);
         dis=dis+(head-curr);
         head=curr;
      }
      dir="right";
   }
   else if(dir=="right")
   {
      for(int i=0;i<right.size();i++)
      {
         curr=right[i];
         track.push_back(curr);
         dis=dis+(curr-head);
         head=curr;
      }
      dir="left";
   }
}
cout<<"Number of seek operations:"<<dis<<endl;
cout<<"Sequence travelled:";
for(int i=0;i<track.size();i++)
```

```
    {
        cout<<track[i]<<" ";
    }
}

int main()
{
    int n;
    cout<<"Enter the number of discs:";
    cin>>n;
    int a[n];
    cout<<"Enter the sequence of discs:";
    for(int i=0;i<n;i++)
    {
        cin>>a[i];
    }
    int head;
    cout<<"Enter the initial head position:";
    cin>>head;
    string dir;
    cout<<"Enter the direction to be traversed from head(left/right):";
    cin>>dir;
    sort(a,a+n);
    scan(a,n,head,dir);
    return 0;
}
```

```
Enter the number of discs:8
Enter the sequence of discs:176 79 34 60 92 11 41 114
Enter the initial head position:50
Enter the direction to be traversed from head(left/right):left
Number of seek operations:226
Sequence travelled:41 34 11 0 60 79 92 114 176

...Program finished with exit code 0
Press ENTER to exit console.
```

16.CSCAN ALGORITHM

```
#include <bits/stdc++.h>
using namespace std;

void scan(int a[],int n,int head,string dir)
{
```

```
    vector<int>left,right,track;
    int dis=0,curr;
    if(dir=="left")
        left.push_back(0);
    else if(dir=="right")
        right.push_back(a[n-1]+1);
    for(int i=0;i<n;i++)
    {
        if(a[i]>head)
            right.push_back(a[i]);
        else if(a[i]<head)
            left.push_back(a[i]);
    }
    sort(left.begin(),left.end());
    sort(right.begin(),right.end());
    for(int i=0;i<right.size();i++)
    {
        curr=right[i];
        track.push_back(curr);
        dis=dis+(curr-head);
        head=curr;
    }
    head=0;
    dis=dis+(a[n-1]+1);
    for(int i=left.size()-1;i>=0;i--)
    {
        curr=left[i];
        track.push_back(curr);
        dis=dis+(head-curr);
        head=curr;
    }
    cout<<"Number of seek operations:"<<dis<<endl;
    cout<<"Sequence travelled:";
    for(int i=0;i<track.size();i++)
    {
        cout<<track[i]<<" ";
    }
}

int main()
```

```
{
   int n;
   cout<<"Enter the number of discs:";
   cin>>n;
   int a[n];
   cout<<"Enter the sequence of discs:";
   for(int i=0;i<n;i++)
   {
      cin>>a[i];
   }
   int head;
   cout<<"Enter the initial head position:";
   cin>>head;
   string dir;
   cout<<"Enter the direction to be traversed from head(left/right):";
   cin>>dir;
   sort(a,a+n);
   scan(a,n,head,dir);
   return 0;
}
```

```
Enter the number of discs:8
Enter the sequence of discs:176 79 34 60 92 11 41 114
Enter the initial head position:50
Enter the direction to be traversed from head(left/right):right
Number of seek operations:293
Sequence travelled:60 79 92 114 176 177 41 34 11

...Program finished with exit code 0
Press ENTER to exit console.
```

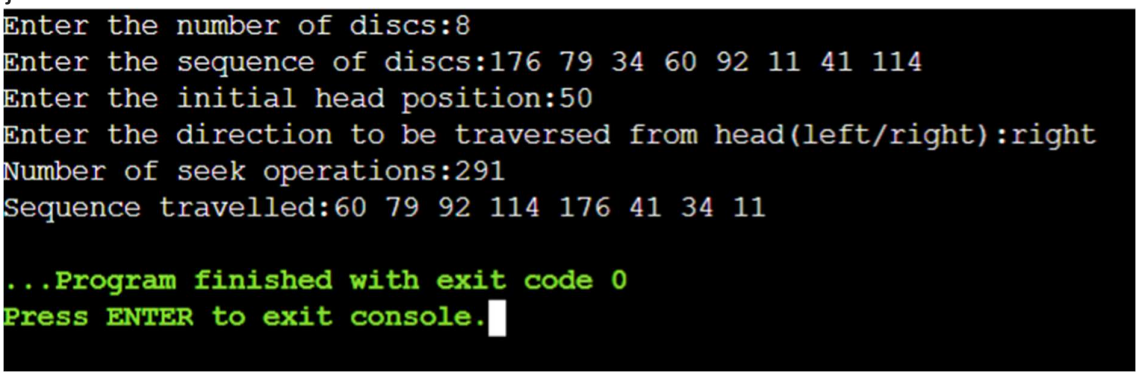
17.LOOK ALGORITHM

```
#include <bits/stdc++.h>
using namespace std;

void scan(int a[],int n,int head,string dir)
{
   vector<int>left,right,track;
   int dis=0,curr;
   for(int i=0;i<n;i++)
   {
```

```
    if(a[i]>head)
      right.push_back(a[i]);
    else if(a[i]<head)
      left.push_back(a[i]);
  }
  sort(left.begin(),left.end());
  sort(right.begin(),right.end());
  int t=2;
  while(t--)
  {
    if(dir=="left")
    {
      for(int i=left.size()-1;i>=0;i--)
      {
        curr=left[i];
        track.push_back(curr);
        dis=dis+(head-curr);
        head=curr;
      }
      dir="right";
    }
    else if(dir=="right")
    {
      for(int i=0;i<right.size();i++)
      {
        curr=right[i];
        track.push_back(curr);
        dis=dis+(curr-head);
        head=curr;
      }
      dir="left";
    }
  }
  cout<<"Number of seek operations:"<<dis<<endl;
  cout<<"Sequence travelled:";
  for(int i=0;i<track.size();i++)
  {
    cout<<track[i]<<" ";
  }
}
```

```
int main()
{
   int n;
   cout<<"Enter the number of discs:";
   cin>>n;
   int a[n];
   cout<<"Enter the sequence of discs:";
   for(int i=0;i<n;i++)
   {
      cin>>a[i];
   }
   int head;
   cout<<"Enter the initial head position:";
   cin>>head;
   string dir;
   cout<<"Enter the direction to be traversed from head(left/right):";
   cin>>dir;
   sort(a,a+n);
   scan(a,n,head,dir);
   return 0;
}
```



*DEADLOCK DETECTION AND AVOIDANCE

18. Bankers algorithm

```
#include<bits/stdc++.h>
using namespace std;
```

```
bool check(vector<int> need,vector<int> available,int r)
{
   for(int i=0;i<r;i++)
   {
      if(need[i]>available[i])
         return false;
   }
   return true;
}
void add(vector<int>allocation,vector<int> &available,int r)
{
   for(int i=0;i<r;i++)
   {
      available[i]+=allocation[i];
   }
}
int main(){
   int p;
   cout<<"Enter the number of processes: ";
   cin>>p;
   int r;
   cout<<"Enter the number of resources: ";
   cin>>r;
   vector<vector<int>>allocation(p,vector<int>(r)),max(p,vector<int>(r));
   cout<<"Enter the allocation data"<<endl;
   for(int i=0;i<p;i++)
   {
      cout<<"Process"<<i<<": ";
      for(int j=0;j<r;j++)
         cin>>allocation[i][j];
   }
   cout<<"Enter the max data"<<endl;
   for(int i=0;i<p;i++)
   {
      cout<<"Process"<<i<<": ";
      for(int j=0;j<r;j++)
         cin>>max[i][j];
   }
   vector<int> available(3);
   cout<<"Enter the available data: ";
```

```
for(int i=0;i<r;i++)
   cin>>available[i];
vector<vector<int>>need(p, vector<int>(r));
for(int i=0;i<p;i++)
{
   for(int j=0;j<r;j++)
   {
      need[i][j]=max[i][j]-allocation[i][j];
   }
}
cout<<endl;
vector<bool>process(p,false);
vector<int>seq;
int pre=1,c=0;
while(pre)
{
   int pre=0;
   for(int i=0;i<p;i++)
   {
      if(process[i]==false && check(need[i],available,r))
      {
         add(allocation[i],available,r);
         process[i]=true;
         seq.push_back(i);
         pre=1;
         c++;
      }
   }
   if(pre==0)
      break;
}
if(c==p)
{
   cout<<"No Deadlock detected"<<endl;
   cout<<"Sequence of process execution: ";
   for(int i=0;i<seq.size()-1;i++)
   {
      cout<<"P"<<seq[i]<<"->";
   }
   cout<<"P"<<seq[seq.size()-1]<<endl;
```

```
    }
    else
        cout<<"Deadlock detected"<<endl;
}
```

```
Enter the number of processes: 5
Enter the number of resources: 3
Enter the allocation data
Process0: 0 1 0
Process1: 2 0 0
Process2: 3 0 2
Process3: 2 1 1
Process4: 0 0 2
Enter the max data
Process0: 7 5 3
Process1: 3 2 2
Process2: 9 0 2
Process3: 2 2 2
Process4: 4 3 3
Enter the available data: 3 3 2

No Deadlock detected
Sequence of process execution: P1->P3->P4->P0->P2
```

19.Resource Request Algorithm

```
#include<bits/stdc++.h>
using namespace std;
bool check(vector<int> request,vector<int> available,int r)
{
    for(int i=0;i<r;i++)
    {
        if(request[i]>available[i])
            return false;
    }
    return true;
}
void add(vector<int>allocation,vector<int> &available,int r)
{
    for(int i=0;i<r;i++)
    {
        available[i]+=allocation[i];
    }
}
```

```
int main(){
   int p;
   cout<<"Enter the number of processes: ";
   cin>>p;
   int r;
   cout<<"Enter the number of resources: ";
   cin>>r;
   vector<vector<int>>allocation(p,vector<int>(r)),request(p,vector<int>(r));
   cout<<"Enter the allocation data"<<endl;
   for(int i=0;i<p;i++)
   {
      cout<<"Process"<<i<<": ";
      for(int j=0;j<r;j++)
         cin>>allocation[i][j];
   }
   cout<<"Enter the request data"<<endl;
   for(int i=0;i<p;i++)
   {
      cout<<"Process"<<i<<": ";
      for(int j=0;j<r;j++)
         cin>>request[i][j];
   }
   vector<int> available(3);
   cout<<"Enter the available data: ";
   for(int i=0;i<r;i++)
      cin>>available[i];
   cout<<endl;
   vector<bool>process(p,false);
   vector<int>seq;
   int pre=1,c=0;
   while(pre)
   {
      int pre=0;
      for(int i=0;i<p;i++)
      {
         if(process[i]==false && check(request[i],available,r))
         {
            add(allocation[i],available,r);
            process[i]=true;
            seq.push_back(i);
```

```
            pre=1;
            c++;
          }
        }
        if(pre==0)
          break;
    }
    if(c==p)
    {
        cout<<"No Deadlock detected"<<endl;
        cout<<"Sequence of process execution: ";
        for(int i=0;i<seq.size()-1;i++)
        {
          cout<<"P"<<seq[i]<<"->";
        }
        cout<<"P"<<seq[seq.size()-1]<<endl;
    }
    else
        cout<<"Deadlock detected"<<endl;
}
```

```
Enter the number of processes: 5
Enter the number of resources: 3
Enter the allocation data
Process0: 0 1 0
Process1: 2 0 0
Process2: 3 0 3
Process3: 2 1 1
Process4: 0 0 2
Enter the request data
Process0: 0 0 0
Process1: 2 0 2
Process2: 0 0 0
Process3: 1 0 0
Process4: 0 0 2
Enter the available data: 0 0 0

No Deadlock detected
Sequence of process execution: P0->P2->P3->P4->P1
```

*FILE ALLOCATION STRATEGIES

20.Sequential algorithm

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
   int total_memory,size_of_each_block,number_of_blocks;
   cout<<"Enter total memory available:";
   cin>>total_memory;
   cout<<"Enter size of each block:";
   cin>>size_of_each_block;

number_of_blocks=ceil(double(total_memory)/double(size_of_each_block));
   int no_of_files;
   cout<<"Enter the number of files:";
   cin>>no_of_files;
   while(no_of_files--)
   {
      string filename;
      cout<<"Enter filename:";
      cin>>filename; int filesize;
      cout<<"Enter filesize:";cin>>filesize;
      int start_block;
      cout<<"Enter start_block:";
      cin>>start_block;
      int
blocks_to_be_allocated=ceil(double(filesize)/double(size_of_each_block));
      vector<int>allocated;
      int temp = start_block;
      for(int i=start_block;i<temp+blocks_to_be_allocated &&
i<number_of_blocks;i++){
         allocated.push_back(i);
      }
      if(start_block+blocks_to_be_allocated>number_of_blocks){
         cout<<(blocks_to_be_allocated-(number_of_blocks-
start_block))<<"blockscouldn't be allocated\n";
      }
      cout<<"Filename\tFilesize\tTotal blocks allocated\tStart Block\tAllocated
blocks\n";
```

```
cout<<filename<<"\t\t"<<filesize<<"\t\t"<<blocks_to_be_allocated<<"\t\t\t"<<
start_block<<"\t\t";
      for(auto x:allocated){
         cout<<x<<" ";
      }
      cout<<endl;
   }
   return 0;
}
```

```
Enter total memory available:1000
Enter size of each block:100
Enter the number of files:1
Enter filename:file1
Enter filesize:200
Enter start_block:5
Filename        Filesize        Total blocks allocated  Start Block     Allocated blocks
file1           200             2                       5               5 6


...Program finished with exit code 0
Press ENTER to exit console.
```

21. Indexed algorithm

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
   int total_memory,size_of_each_block,number_of_blocks;
   cout<<"Enter total memory available:";
   cin>>total_memory;
   cout<<"Enter size of each block:";
   cin>>size_of_each_block;

number_of_blocks=ceil(double(total_memory)/double(size_of_each_block));
   int no_of_files;
   cout<<"Enter the number of files:";
   cin>>no_of_files;
   while(no_of_files--)
```

```
   {
      string filename;
      cout<<"Enter filename:";
      cin>>filename;
      int filesize;
      cout<<"Enter filesize:";
      cin>>filesize;
      int
blocks_to_be_allocated=ceil(double(filesize)/double(size_of_each_block));
      int index_of_data_blocks;
      cout<<"Enter the index of the data blocks:";
      cin>>index_of_data_blocks;
      cout<<"Enter the data blocks to be allocated:";
      vector<int>allocated(blocks_to_be_allocated);
      for(int i=0;i<blocks_to_be_allocated;i++)
         cin>>allocated[i];
      cout<<"Filename\tFilesize\tTotal blocks allocated\tAllocated blocks\n";

cout<<filename<<"\t\t"<<filesize<<"\t\t"<<blocks_to_be_allocated<<"\t\t\t";
      sort(allocated.begin(),allocated.end());
      for(auto x:allocated){
         cout<<x<<" ";
      }
      cout<<endl;
   }
   return 0;
}
```

```
Enter total memory available:1000
Enter size of each block:100
Enter the number of files:1
Enter filename:file2
Enter filesize:300
Enter the index of the data blocks:3
Enter the data blocks to be allocated:5 2 8
Filename        Filesize        Total blocks allocated  Allocated blocks
file2           300             3                       2 5 8
```

22. Linked algorithm

```
#include<bits/stdc++.h>
using namespace std;
struct Node{
int data;
struct Node* next;
};
int main()
{
   int total_memory,size_of_each_block,number_of_blocks;
   cout<<"Enter total memory available:";
   cin>>total_memory;
   cout<<"Enter size of each block:";
   cin>>size_of_each_block;

number_of_blocks=ceil(double(total_memory)/double(size_of_each_block));
   int no_of_files;
   cout<<"Enter the number of files:";
   cin>>no_of_files;
   while(no_of_files--)
   {
      string filename;
      cout<<"Enter filename:";
      cin>>filename;
      int filesize;
      cout<<"Enter filesize:";
      cin>>filesize;
      int
blocks_to_be_allocated=ceil(double(filesize)/double(size_of_each_block-1));
      struct Node* head=new Node();
      struct Node* temp =new Node();
      int i=0;
      cout<<"Enter the data blocks:";
      while(i<blocks_to_be_allocated)
      {
         int x;
         cin>>x;
         if(i==0){
            temp->data = x;
            temp->next = NULL;
```

```
            head = temp;
         }
         else{
            struct Node* nn=new Node();
            temp->next=nn;
            nn->data=x;
            nn->next=NULL;
            temp=nn;
            if(i==1) head->next=temp;
         }
         i++;
      }
      cout<<"Filename\tFilesize\tTotal blocks allocated\tAllocated blocks\n";

cout<<filename<<"\t\t"<<filesize<<"\t\t"<<blocks_to_be_allocated<<"\t\t\t";
      while(head!=NULL)
      {
         cout<<head->data<<" ";
         head=head->next;
      }
      cout<<endl;
   }
   return 0;
}
```

```
Enter total memory available:1000
Enter size of each block:100
Enter the number of files:1
Enter filename:file3
Enter filesize:200
Enter the data blocks:5 6 7
Filename        Filesize        Total blocks allocated  Allocated blocks
file3           200             3                       5 6 7
```