**WAP IN C TO SIMULATE FCFS CPU SCHEDULING ALGORITHM**

#include<stdio.h>

int main()

{

int p[10],at[10],bt[10],ct[10],tat[10],wt[10],i,j,temp=0,n;

float awt=0,atat=0;

printf("enter no of process you want:");

scanf("%d",&n);

printf("enter %d process:",n);

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

{

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

}

printf("enter %d arrival time:",n);

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

{

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

}

printf ("enter %d burst time:",n);

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

{

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

}

// sorting at,bt, and process according to at

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

{

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

{

if(at[j]>at[j+1])

{

temp=p[j+1];

p[j+1]=p[j];

p[j]=temp;

temp=at[j+1];

at[j+1]=at[j];

at[j]=temp;

temp=bt[j+1];

bt[j+1]=bt[j];

bt[j]=temp;

}

}

}

/\* calculating 1st ct \*/

ct[0]=at[0]+bt[0];

/\* calculating 2 to n ct \*/

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

{

//when proess is ideal in between i and i+1

temp=0;

if(ct[i-1]<at[i])

{

temp=at[i]-ct[i-1];

}

ct[i]=ct[i-1]+bt[i]+temp;

}

/\* calculating tat and wt \*/

printf("\np\t A.T\t B.T\t C.T\t TAT\t WT");

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

{

tat[i]=ct[i]-at[i];

wt[i]=tat[i]-bt[i];

atat+=tat[i];

awt+=wt[i];

}

atat=atat/n;

awt=awt/n;

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

{

printf("\nP%d\t %d\t %d\t %d \t %d \t %d",p[i],at[i],bt[i],ct[i],tat[i],wt[i]);

}

printf("\naverage turnaround time is %f",atat);

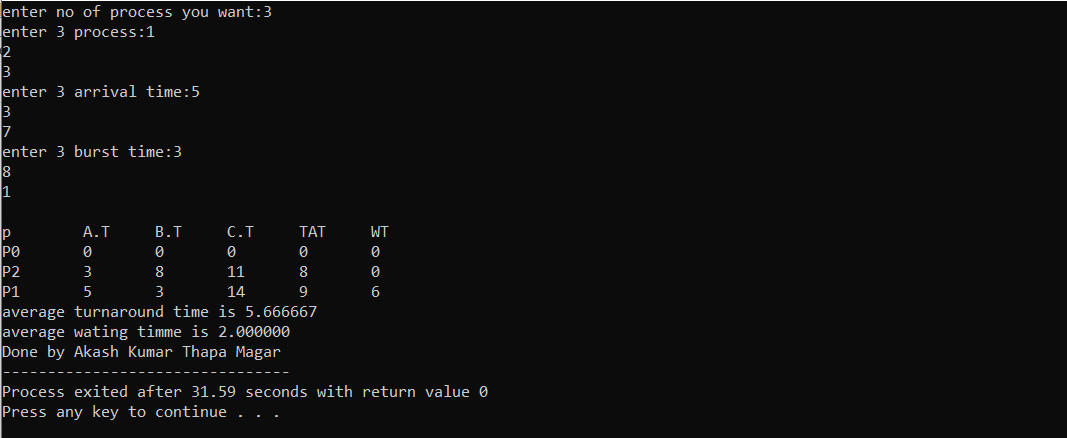
printf("\naverage wating timme is %f",awt);

printf("\nDone by Akash Kumar Thapa Magar");

return 0;

}

**OUTPUT:**



**WAP IN C TO SIMULATE SJF CPU SCHEDULING ALGORITHM**

#include<stdio.h>

int main()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;

float avg\_wt,avg\_tat;

printf("Enter number of process:");

scanf("%d",&n);

printf("\nEnter Burst Time:\n");

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

{

printf("p%d:",i+1);

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

p[i]=i+1;

}

//sorting of burst times

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

{

pos=i;

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

{

if(bt[j]<bt[pos])

pos=j;

}

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0;

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

{

wt[i]=0;

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

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=(float)total/n;

total=0;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");

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

{

tat[i]=bt[i]+wt[i];

total+=tat[i];

printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

avg\_tat=(float)total/n;

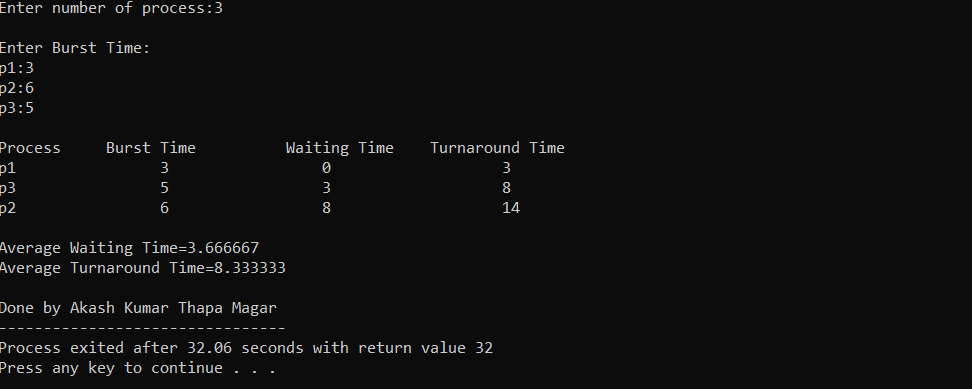
printf("\n\nAverage Waiting Time=%f",avg\_wt);

printf("\nAverage Turnaround Time=%f\n",avg\_tat);

printf("\nDone by Akash Kumar Thapa Magar");

}

**OUTPUT:**



**WAP IN C TO SIMULATE SRTF CPU SCHEDULING ALGORITHM**

#include <stdio.h>

int main()

{

int arrival\_time[10], burst\_time[10], temp[10];

int i, smallest, count = 0, time, limit;

double wait\_time = 0, turnaround\_time = 0, end;

float average\_waiting\_time, average\_turnaround\_time;

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

scanf("%d", &limit);

printf("\nEnter Details of %d Processesn", limit);

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

{

printf("\nEnter Arrival Time:\t");

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

printf("Enter Burst Time:\t");

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

temp[i] = burst\_time[i];

}

burst\_time[9] = 9999;

for(time = 0; count != limit; time++)

{

smallest = 9;

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

{

if(arrival\_time[i] <= time && burst\_time[i] < burst\_time[smallest] && burst\_time[i] > 0)

{

smallest = i;

}

}

burst\_time[smallest]--;

if(burst\_time[smallest] == 0)

{

count++;

end = time + 1;

wait\_time = wait\_time + end - arrival\_time[smallest] - temp[smallest];

turnaround\_time = turnaround\_time + end - arrival\_time[smallest];

}

}

average\_waiting\_time = wait\_time / limit;

average\_turnaround\_time = turnaround\_time / limit;

printf("\n\nAverage Waiting Time:\t%lf\n", average\_waiting\_time);

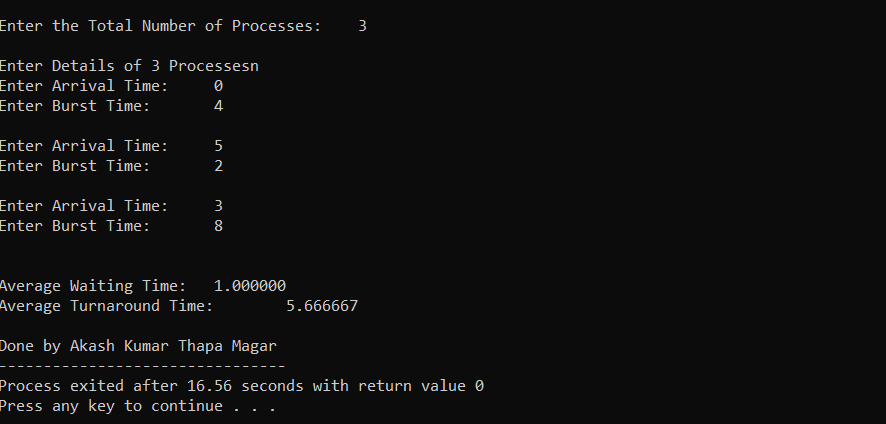
printf("Average Turnaround Time:\t%lf\n", average\_turnaround\_time);

printf("\nDone by Akash Kumar Thapa Magar");

return 0;

}

**OUTPUT:**



**WAP IN C TO SIMULATE ROUND ROBIN CPU SCHEDULING ALGORITHM**

#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);

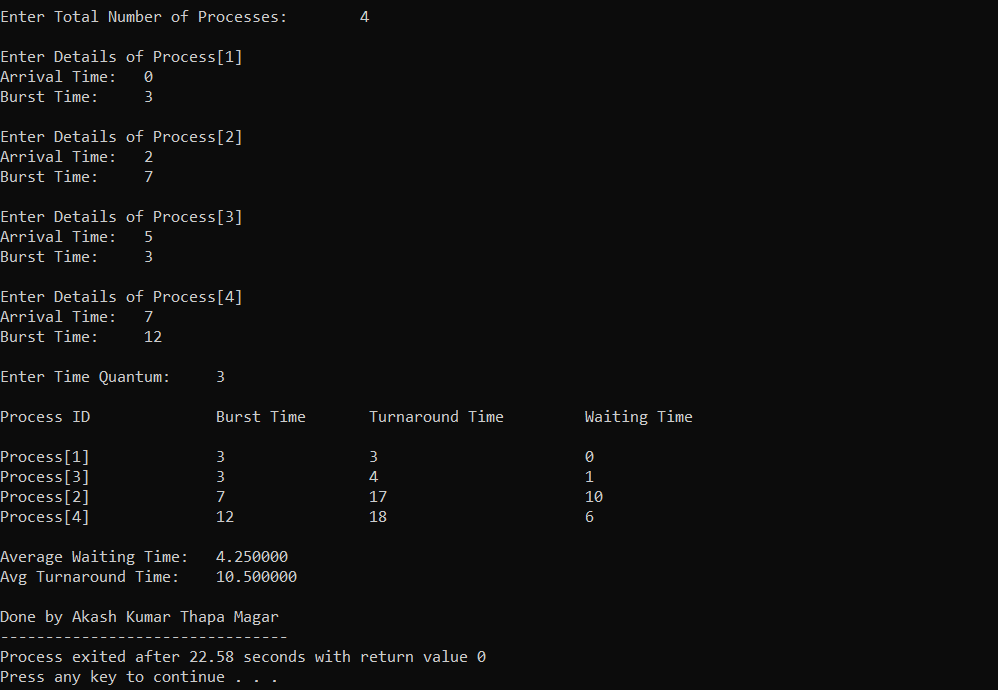
printf("\nAvg Turnaround Time:\t%f\n", average\_turnaround\_time);

printf("\nDone by Akash Kumar Thapa Magar");

return 0;

}

**OUTPUT:**



**WAP IN C TO SIMULATE NON-PREEMPTIVE PRIORITY SCHEDULING ALGORITHM**

#include<stdio.h>

struct process

{

int id,WT,AT,BT,TAT,PR;

};

struct process a[10];

void swap(int \*b,int \*c)

{

int tem;

tem=\*c;

\*c=\*b;

\*b=tem;

}

int main()

{

int n,check\_ar=0;

int Cmp\_time=0;

float Total\_WT=0,Total\_TAT=0,Avg\_WT,Avg\_TAT;

printf("Enter the number of process \n");

scanf("%d",&n);

printf("Enter the Arrival time , Burst time and priority of the process\n");

printf("AT BT PR\n");

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

{

scanf("%d%d%d",&a[i].AT,&a[i].BT,&a[i].PR);

a[i].id=i+1;

if(i==0)

check\_ar=a[i].AT;

if(check\_ar!=a[i].AT )

check\_ar=1;

}

if(check\_ar!=0)

{

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

{

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

{

if(a[j].AT>a[j+1].AT)

{

swap(&a[j].id,&a[j+1].id);

swap(&a[j].AT,&a[j+1].AT);

swap(&a[j].BT,&a[j+1].BT);

swap(&a[j].PR,&a[j+1].PR);

}

}

}

}

if(check\_ar!=0)

{

a[0].WT=a[0].AT;

a[0].TAT=a[0].BT-a[0].AT;

Cmp\_time=a[0].TAT;

Total\_WT=Total\_WT+a[0].WT;

Total\_TAT=Total\_TAT+a[0].TAT;

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

{

int min=a[i].PR;

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

{

if(min>a[j].PR && a[j].AT<=Cmp\_time)

{

min=a[j].PR;

swap(&a[i].id,&a[j].id);

swap(&a[i].AT,&a[j].AT);

swap(&a[i].BT,&a[j].BT);

swap(&a[i].PR,&a[j].PR);

}

}

a[i].WT=Cmp\_time-a[i].AT;

Total\_WT=Total\_WT+a[i].WT;

Cmp\_time=Cmp\_time+a[i].BT;

a[i].TAT=Cmp\_time-a[i].AT;

Total\_TAT=Total\_TAT+a[i].TAT;

}

}

else

{

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

{

int min=a[i].PR;

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

{

if(min>a[j].PR && a[j].AT<=Cmp\_time)

{

min=a[j].PR;

swap(&a[i].id,&a[j].id);

swap(&a[i].AT,&a[j].AT);

swap(&a[i].BT,&a[j].BT);

swap(&a[i].PR,&a[j].PR);

}

}

a[i].WT=Cmp\_time-a[i].AT;

Cmp\_time=Cmp\_time+a[i].BT;

a[i].TAT=Cmp\_time-a[i].AT;

Total\_WT=Total\_WT+a[i].WT;

Total\_TAT=Total\_TAT+a[i].TAT;

}

}

Avg\_WT=Total\_WT/n;

Avg\_TAT=Total\_TAT/n;

printf("The process are\n");

printf("ID WT TAT\n");

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

{

printf("%d\t%d\t%d\n",a[i].id,a[i].WT,a[i].TAT);

}

printf("Avg waiting time is: %f\n",Avg\_WT);

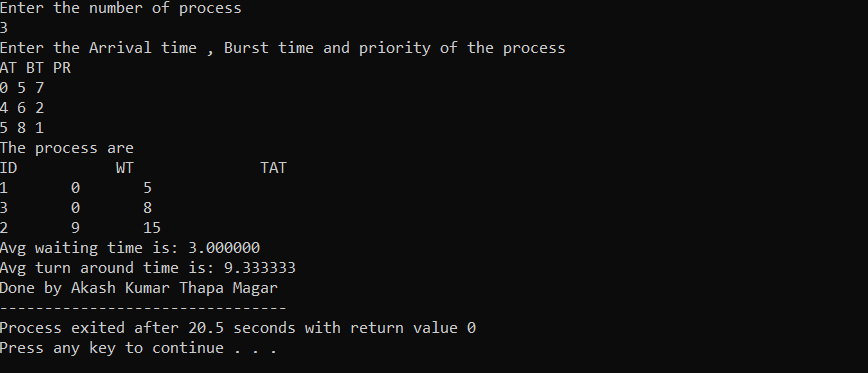
printf("Avg turn around time is: %f",Avg\_TAT);

printf("\nDone by Akash Kumar Thapa Magar");

return 0;

}

**OUTPUT:**



**WAP IN C TO SIMULATE PREEMPTIVE PRIORITY SCHEDULING ALGORITHM**

#include<stdio.h>

struct process

{

int WT,AT,BT,TAT,PT;

};

struct process a[10];

int main()

{

int n,temp[10],t,count=0,short\_p;

float total\_WT=0,total\_TAT=0,Avg\_WT,Avg\_TAT;

printf("Enter the number of the process\n");

scanf("%d",&n);

printf("Enter the arrival time , burst time and priority of the process\n");

printf("AT BT PT\n");

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

{

scanf("%d%d%d",&a[i].AT,&a[i].BT,&a[i].PT);

temp[i]=a[i].BT;

}

a[9].PT=10000;

for(t=0;count!=n;t++)

{

short\_p=9;

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

{

if(a[short\_p].PT>a[i].PT && a[i].AT<=t && a[i].BT>0)

{

short\_p=i;

}

}

a[short\_p].BT=a[short\_p].BT-1;

if(a[short\_p].BT==0)

{

count++;

a[short\_p].WT=t+1-a[short\_p].AT-temp[short\_p];

a[short\_p].TAT=t+1-a[short\_p].AT;

total\_WT=total\_WT+a[short\_p].WT;

total\_TAT=total\_TAT+a[short\_p].TAT;

}

}

Avg\_WT=total\_WT/n;

Avg\_TAT=total\_TAT/n;

printf("ID WT TAT\n");

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

{

printf("%d %d\t%d\n",i+1,a[i].WT,a[i].TAT);

}

printf("Avg waiting time of the process is %f\n",Avg\_WT);

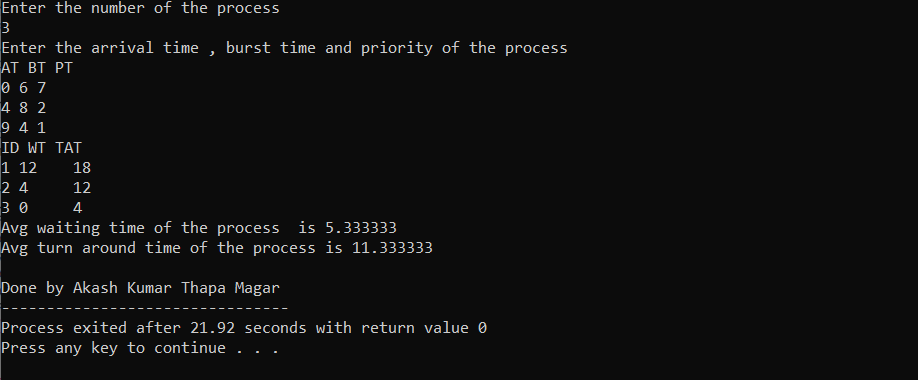
printf("Avg turn around time of the process is %f\n",Avg\_TAT);

printf("\nDone by Akash Kumar Thapa Magar");

return 0;

}

**OUTPUT:**



**WAP TO IMPLEMENT BANKERS ALGORITHM FOR MULTIPLE TYPE OF RESOURCES TO DECIDE SAFE/UNSAFE STATE.**

#include <stdio.h>

int current[5][5], maximum\_claim[5][5], available[5];

int allocation[5] = {0, 0, 0, 0, 0};

int maxres[5], running[5], safe = 0;

int counter = 0, i, j, exec, resources, processes, k = 1;

int main()

{

printf("\nEnter number of processes: ");

scanf("%d", &processes);

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

{

running[i] = 1;

counter++;

}

printf("\nEnter number of resources: ");

scanf("%d", &resources);

printf("\nEnter Claim Vector:");

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

{

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

}

printf("\nEnter Allocated Resource Table:\n");

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

{

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

{

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

}

}

printf("\nEnter Maximum Claim Table:\n");

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

{

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

{

scanf("%d", &maximum\_claim[i][j]);

}

}

printf("\nThe Claim Vector is: ");

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

{

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

}

printf("\nThe Allocated Resource Table:\n");

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

{

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

{

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

}

printf("\n");

}

printf("\nThe Maximum Claim Table:\n");

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

{

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

{

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

}

printf("\n");

}

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

{

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

{

allocation[j] += current[i][j];

}

}

printf("\nAllocated resources:");

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

{

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

}

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

{

available[i] = maxres[i] - allocation[i];

}

printf("\nAvailable resources:");

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

{

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

}

printf("\n");

while (counter != 0)

{

safe = 0;

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

{

if (running[i])

{

exec = 1;

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

{

if (maximum\_claim[i][j] - current[i][j] > available[j])

{

exec = 0;

break;

}

}

if (exec)

{

printf("\nProcess%d is executing\n", i + 1);

running[i] = 0;

counter--;

safe = 1;

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

{

available[j] += current[i][j];

}

break;

}

}

}

if (!safe)

{

printf("\nThe processes are in unsafe state.\n");

break;

}

else

{

printf("\nThe process is in safe state");

printf("\nAvailable vector:");

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

{

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

}

printf("\n");

}

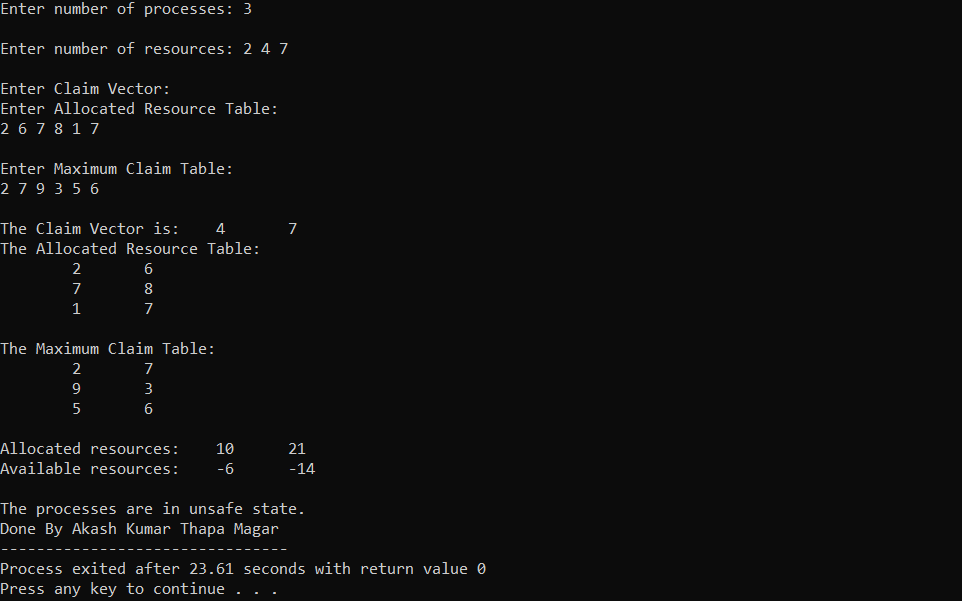
}

printf("Done By Akash Kumar Thapa Magar");

return 0;

}

**OUTPUT:**



**WAP TO IMPLEMENT FIFO PAGE REPLACEMENT ALGORITHM**

#include<iostream>

using namespace std;

int main()

{

int i,j,n,a[50],frame[10],no,k,avail,count=0;

cout<<"Enter the number of pages:";

cin>>n;

cout<<"Enter the page number:"<<endl;

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

{

cin>>a[i];

}

cout<<"Enter the nunmber of frames:";

cin>>no;

for(i=0;i<no;i++)

{

frame[i]=-1;

j=0;

}

cout<<"\n";

cout<<"Ref string\tPage frames\n";

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

{

cout<<a[i]<<"\t\t";

avail=0;

for(k=0;k<no;k++){

if(frame[k]==a[i])

{

avail=1;

}

}

if (avail==0)

{

frame[j]=a[i];

j=(j+1)%no;

count++;

for(k=0;k<no;k++)

{

cout<<frame[k]<<"\t";

}

}

cout<<"\n";

}

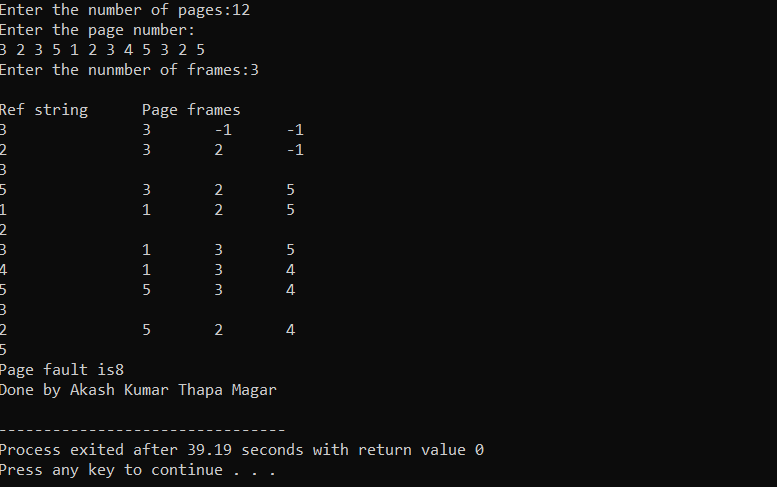
cout<<"Page fault is"<<count<<endl;

cout<<"Done by Akash Kumar Thapa Magar"<<endl;

return 0;

}

**OUTPUT:**



**WAP FOR STIMULATING LEAST FREQUENTLY USED ALGORITHM**

#include<stdio.h>

void print(int frameno,int frame[])

{

int j;

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

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

printf("\n");

}

int main()

{

int i,j,k,n,page[50],frameno,frame[10],move=0,flag,count=0,count1[10]={0},

repindex,leastcount;

float rate;

printf("Enter the number of pages\n");

scanf("%d",&n);

printf("Enter the page reference numbers\n");

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

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

printf("Enter the number of frames\n");

scanf("%d",&frameno);

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

frame[i]=-1;

printf("Page reference string\tFrames\n");

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

{

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

flag=0;

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

{

if(page[i]==frame[j])

{

flag=1;

count1[j]++;

printf("No replacement\n");

break;

}

}

if(flag==0&&count<frameno)

{

frame[move]=page[i];

count1[move]=1;

move=(move+1)%frameno;

count++;

print(frameno,frame);

}

else if(flag==0)

{

repindex=0;

leastcount=count1[0];

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

{

if(count1[j]<leastcount)

{

repindex=j;

leastcount=count1[j];

}

}

frame[repindex]=page[i];

count1[repindex]=1;

count++;

print(frameno,frame);

}

}

rate=(float)count/(float)n;

printf("Number of page faults is %d\n",count);

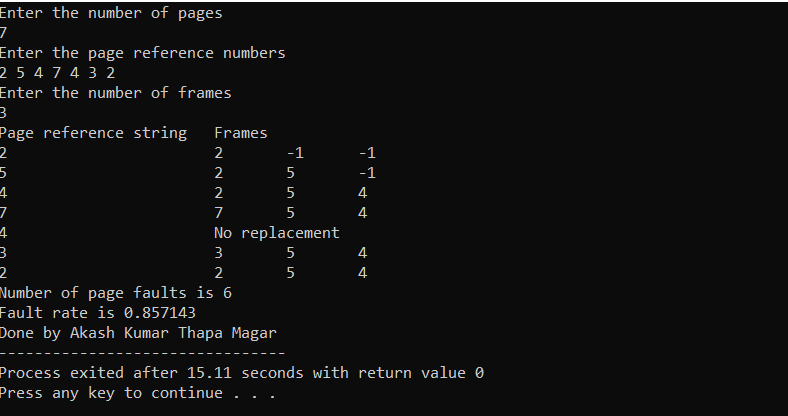
printf("Fault rate is %f\n",rate);

printf("Done by Akash Kumar Thapa Magar");

return 0;

}

**OUTPUT:**



**WAP FOR STIMULATING LEAST RECENTLY USED ALGORITHM**

#include<stdio.h>

int findLRU(int time[], int n){

int i, minimum = time[0], pos = 0;

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

if(time[i] < minimum){

minimum = time[i];

pos = i;

}

}

return pos;

}

int main()

{

int no\_of\_frames, no\_of\_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos, faults = 0;

printf("Enter number of frames: ");

scanf("%d", &no\_of\_frames);

printf("Enter number of pages: ");

scanf("%d", &no\_of\_pages);

printf("Enter reference string: ");

for(i = 0; i < no\_of\_pages; ++i){

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

}

for(i = 0; i < no\_of\_frames; ++i){

frames[i] = -1;

}

for(i = 0; i < no\_of\_pages; ++i){

flag1 = flag2 = 0;

for(j = 0; j < no\_of\_frames; ++j){

if(frames[j] == pages[i]){

counter++;

time[j] = counter;

flag1 = flag2 = 1;

break;

}

}

if(flag1 == 0){

for(j = 0; j < no\_of\_frames; ++j){

if(frames[j] == -1){

counter++;

faults++;

frames[j] = pages[i];

time[j] = counter;

flag2 = 1;

break;

}

}

}

if(flag2 == 0){

pos = findLRU(time, no\_of\_frames);

counter++;

faults++;

frames[pos] = pages[i];

time[pos] = counter;

}

printf("\n");

for(j = 0; j < no\_of\_frames; ++j){

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

}

}

printf("\n\nTotal Page Faults = %d", faults);

printf("\nDone by Akash Kumar Thapa Magar");

return 0;

}

**OUTPUT:**



**WAP FOR STIMULATING OPTIMAL PAGE REPLACEMENT ALGORITHM**

#include<stdio.h>

int main()

{

int no\_of\_frames, no\_of\_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults = 0;

printf("Enter number of frames: ");

scanf("%d", &no\_of\_frames);

printf("Enter number of pages: ");

scanf("%d", &no\_of\_pages);

printf("Enter page reference string: ");

for(i = 0; i < no\_of\_pages; ++i){

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

}

for(i = 0; i < no\_of\_frames; ++i){

frames[i] = -1;

}

for(i = 0; i < no\_of\_pages; ++i){

flag1 = flag2 = 0;

for(j = 0; j < no\_of\_frames; ++j){

if(frames[j] == pages[i]){

flag1 = flag2 = 1;

break;

}

}

if(flag1 == 0){

for(j = 0; j < no\_of\_frames; ++j){

if(frames[j] == -1){

faults++;

frames[j] = pages[i];

flag2 = 1;

break;

}

}

}

if(flag2 == 0){

flag3 =0;

for(j = 0; j < no\_of\_frames; ++j){

temp[j] = -1;

for(k = i + 1; k < no\_of\_pages; ++k){

if(frames[j] == pages[k]){

temp[j] = k;

break;

}

}

}

for(j = 0; j < no\_of\_frames; ++j){

if(temp[j] == -1){

pos = j;

flag3 = 1;

break;

}

}

if(flag3 ==0){

max = temp[0];

pos = 0;

for(j = 1; j < no\_of\_frames; ++j){

if(temp[j] > max){

max = temp[j];

pos = j;

}

}

}

frames[pos] = pages[i];

faults++;

}

printf("\n");

for(j = 0; j < no\_of\_frames; ++j){

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

}

}

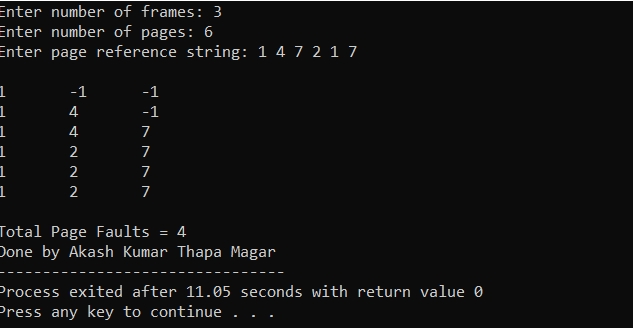
printf("\n\nTotal Page Faults = %d", faults);

printf("\nDone by Akash Kumar Thapa Magar");

return 0;

}

**OUTPUT:**



**WAP FOR STIMULATING CLOCK REPLACEMENT ALGORITHM**

#include<stdio.h>

int main()

{

int n,p[100],f[10],ava,hit=0,usebit[10],i,j;

printf("enter the length of the Reference string: ");

scanf("%d",&n);

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

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

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

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

{

ava=0;

// found

for(j=0;j<3;j++)

{

if(p[i]==f[j])

{

ava=1;

hit++;

usebit[j]=1;

break;

}

}

//search for usebit 0

if(ava==0)

{

for(j=0;j<3;j++)

{

if(usebit[j]==0)

{

f[j]=p[i];

usebit[j]=1;

ava=1;

break;

}

}

}

// fifo

if(ava==0)

{

for(j=0;j<3;j++)

usebit[j]=0;

}

f[0]=p[i];

usebit[0]=1;

}

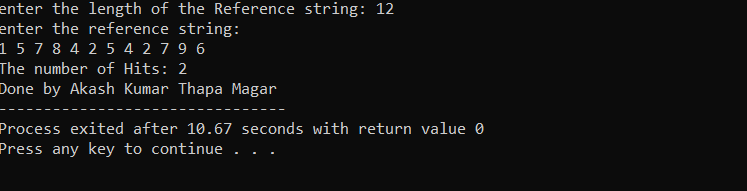
printf("The number of Hits: %d",hit);

printf("\nDone by Akash Kumar Thapa Magar");

return 0;

}

**OUTPUT:**



**WAP FOR SIMULATING SECOND CHANCE PAGE REPLACEMENT**

#include<iostream>

using namespace std;

#define size 3

int full=0;

int a[30];

int ref[size];

int frame[size];

int repptr=0;

int count=0;

int display(){

int i;

cout<<"The elements in the frame are:"<<endl;

for(i=0;i<full;i++){

cout<<frame[i]<<"\n";

}

}

int pagerep(int ele){

int temp;

while(ref[repptr]!=0){

ref[repptr++]=0;

if(repptr==size){

repptr=0;

}

}

temp=frame[repptr];

frame[repptr]=ele;

ref[repptr]=1;

return temp;

}

int pagefault(int ele){

if(full !=size)

ref[full]=1;

frame[full++]=ele;

}else{

cout<<"The page replaced is "<<pagerep(ele)<<endl;

}

}

int search(int ele){

int i, flag;

flag=0;

if(full!=0){

for(i=0;i<full;i++){

if(ele==frame[i]){

flag=1;

ref[i]=1;

break;

}

}

}

return flag;

}

int main(){

int n,i;

cout<<"Enter the number of elements in the reference string are:";

cin>>n;

cout<<"Enter elements in the string "<<endl;

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

cin>>a[i];

}

cout<<"\n";

cout<<"Elements present in the string are:"<<endl;

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

cout<<a[i]<<"\t";

}

cout<<"\n";

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

if(search(a[i])!=1){

pagefault(a[i]);

display();

count ++;

}

}

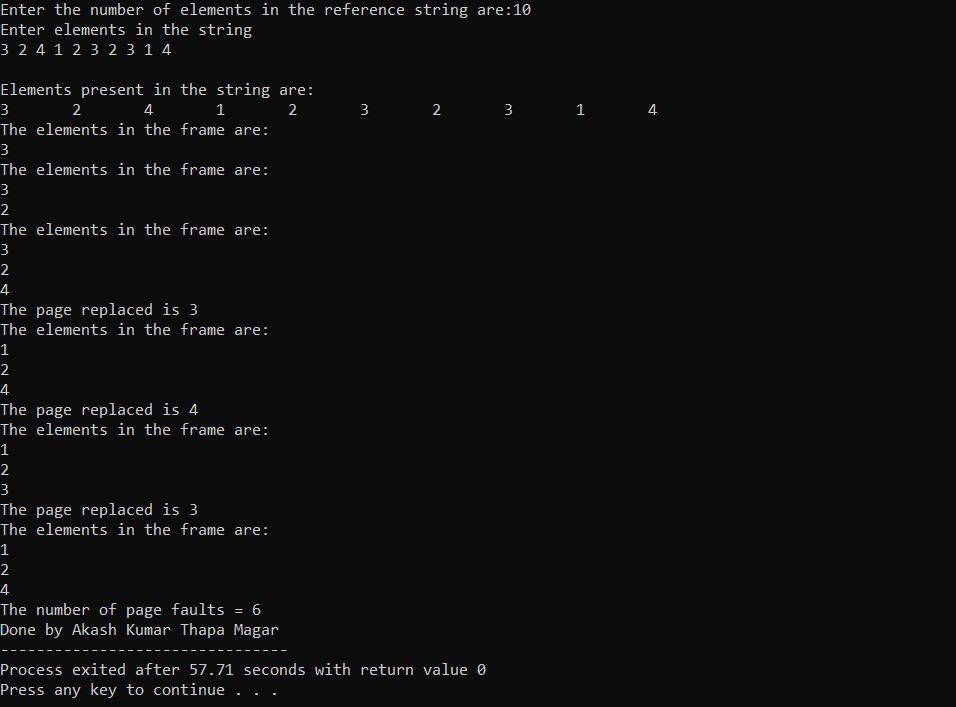
cout<<"The number of page faults = "<<count<<endl;

cout<<"Done by Akash Kumar Thapa Magar";

return 0;

}

**OUTPUT:**



**WAP TO STIMULATE SHORTEST SEEK TIME FIRST DISK SCHEDULING ALGORITHM**

#include<iostream>

using namespace std;

struct head{

int num,flag;

};

int main(){

struct head h[50];

int a[50],b[50];

int count=0,j,x,limit,min,location,disk\_head,sum=0;

cout<<"Enter total no. of locations:";

cin>>limit;

cout<<"Enter position of disk head:";

cin>>disk\_head;

cout<<"Enter elements of disk head queue"<<endl;

while(count<limit){

cin>>h[count].num;

h[count].flag=0;

count++;

}

for(count=0;count<limit;count++)

{

x=0;

min=0;

location=0;

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

{

if(h[j].flag==0){

if(x==0){

a[j]=disk\_head-h[j].num;

if(a[j]<0){

a[j]=h[j].num-disk\_head;

}

min=a[j];

location=j;

x++;

}else{

a[j]=disk\_head-h[j].num;

if(a[j]<0)

{

a[j]=h[j].num-disk\_head;

}

}

if(min>a[j]){

min=a[j];

location=j;

}

}

}

h[location].flag=1;

b[count]=h[location].num-disk\_head;

if(b[count]<0)

{

b[count]=disk\_head-h[location].num;

}

disk\_head=h[location].num;

}

count=0;

while(count<limit){

sum=sum+b[count];

count++;

}

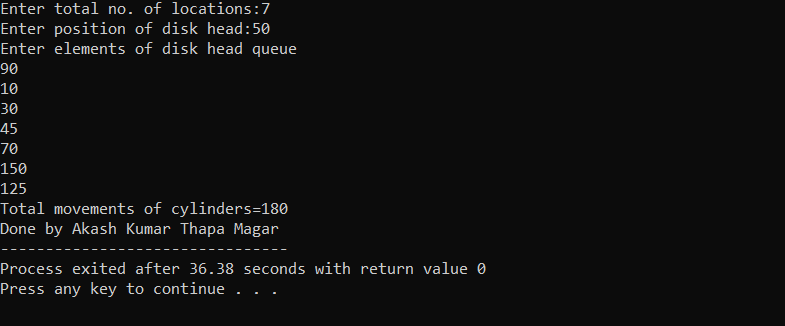
cout<<"Total movements of cylinders="<<sum<<endl;

cout<<"Done by Akash Kumar Thapa Magar";

return 0;

}

**OUTPUT:**



**WAP to SIMULATE FCFS DISK SCHEDULING ALGORITHM**

#include <stdio.h>

int i;

int waitingtime(int proc[], int n,

int burst\_time[], int wait\_time[]) {

wait\_time[0] = 0;

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

wait\_time[i] = burst\_time[i-1] + wait\_time[i-1] ;

return 0;

}

int turnaroundtime( int proc[], int n,

int burst\_time[], int wait\_time[], int tat[]) {

int i;

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

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

return 0;

}

int avgtime( int proc[], int n, int burst\_time[]) {

int wait\_time[n], tat[n], total\_wt = 0, total\_tat = 0;

int i;

waitingtime(proc, n, burst\_time, wait\_time);

turnaroundtime(proc, n, burst\_time, wait\_time, tat);

printf("Processes Burst Waiting Turn around \n");

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

total\_wt = total\_wt + wait\_time[i];

total\_tat = total\_tat + tat[i];

printf(" %d\t %d\t\t %d \t%d\n", i+1, burst\_time[i], wait\_time[i], tat[i]);

}

printf("Average waiting time = %f\n", (float)total\_wt / (float)n);

printf("Average turn around time = %f\n", (float)total\_tat / (float)n);

return 0;

}

int main() {

int proc[] = { 1, 2, 3};

int n = sizeof proc / sizeof proc[0];

int burst\_time[] = {5, 8, 12};

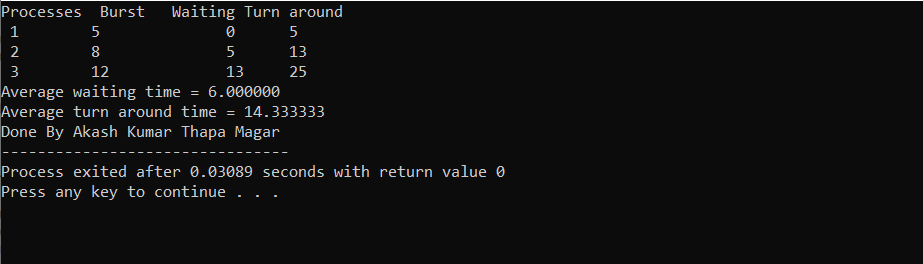
avgtime(proc, n, burst\_time);

printf(“Done By Akash Kumar Thapa Magar”);

return 0;

}

**OUTPUT:**



**WAP to SIMULATE SCAN DISK SCHEDULING ALGORITHM**

#include<iostream>

using namespace std;

void scan\_algorithm(int left[],int right[],int count, int limit){

int arr[50];

int x=count-1,y=count+1,c=0,d=0,j;

while(x>-1){

cout<<"\n x="<<x;

cout<<"\n left["<<x<<"]="<<left[x];

arr[d]=left[x];

x--;

d++;

}

arr[d]=0;

while(y<limit+1){

arr[y]=right[c];

c++;

y++;

}

cout<<"\n scanning order"<<endl;

for(j=0;j<limit+1;j++){

cout<<arr[j]<<"\t";

}}

void division(int elements[], int limit, int disk\_head){

int count=0,p,q,m,x;

int left[50],right[50];

for(count=0;count<limit;count++){

if(elements[count]>disk\_head){

cout<<"\n Break position: "<<elements[count];

break;

}

}

cout<<"\n value: "<<count;

q=1;

p=0;

m=limit;

left[0]=elements[0];

cout<<"\n Left:"<<left[0];

while(q<count){

cout<<"\n element[1] value: "<<elements[q];

left[q]=elements[q];

cout<<"\n left: "<<left[q];

q++;

cout<<"\n |:"<<q;

}

x=count;

while(x<m){

right[p]=elements[x];

cout<<"\n right:"<<right[p];

cout<<"\n elements:"<<elements[x];

p++;

x++;

}

scan\_algorithm(left, right, count, limit);

}

void sorting(int elements[],int limit){

int location,count,j,temp,small;

for(count=0;count<limit-1;count++){

small=elements[count];

location=count;

for(j=count+1;j<limit;j++){

if(small>elements[j]){

small=elements[j];

location=j;

}

}

temp=elements[location];

elements[location]=elements[count];

elements[count]=temp;

}

}

int main(){

int count,disk\_head,elements[40],limit,sum;

cout<<"Enter total no. of location:";

cin>>limit;

cout<<"Enter position of disk head:";

cin>>disk\_head;

cout<<"Enter elements of disk head queue:"<<endl;

for(count=0;count<limit;count++){

cout<<"elements["<<count+1<<"]=";

cin>>elements[count];

}

sorting(elements, limit);

division(elements,limit,disk\_head);

count=0;

while(count<limit){

sum=disk\_head+elements[count];

count++;

}

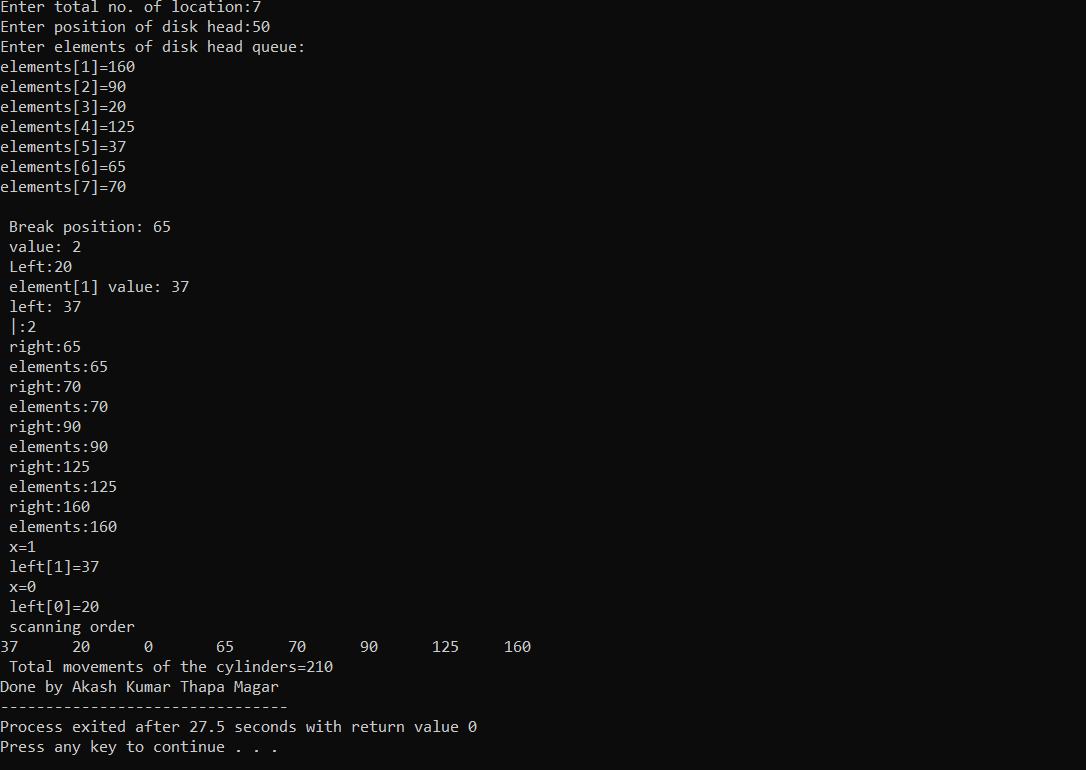
cout<<"\n Total movements of the cylinders="<<sum<<endl;

cout<<"Done by Akash Kumar Thapa Magar";

return 0;

}

**OUTPUT:**



**WAP TO SIMULATE BEST FIT DISK SCHEOULING ALGORITHM**

#include<iostream>

using namespace std;

int main(){

int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;

static int barray[20],parray[20];

cout<<"Enter no. of blocks: ";

cin>>nb;

cout<<"Enter no. of processes: ";

cin>>np;

cout<<"Enter the size of the blocks: "<<endl;

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

cout<<"Block no."<<i<<" =";

cin>>b[i];

}

cout<<"Enter the size of the processes: "<<endl;

for(i=1;i<=np;i++){

cout<<"Process no."<<i<<" =";

cin>>p[i];

}

for(i=1;i<=np;i++){

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

if(barray[j]!=1){

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

if(temp>=0){

if(lowest>temp){

parray[i]=j;

lowest=temp;

}

}

}

}

fragment[i]=lowest;

barray[parray[i]]=1;

lowest=10000;

}

cout<<"\nProcess\_no \tProcess\_size \tBlock\_no \tBlock\_size \tFragment:"<<endl;

for(i=1;i<=np && parray[i]!=0;i++){

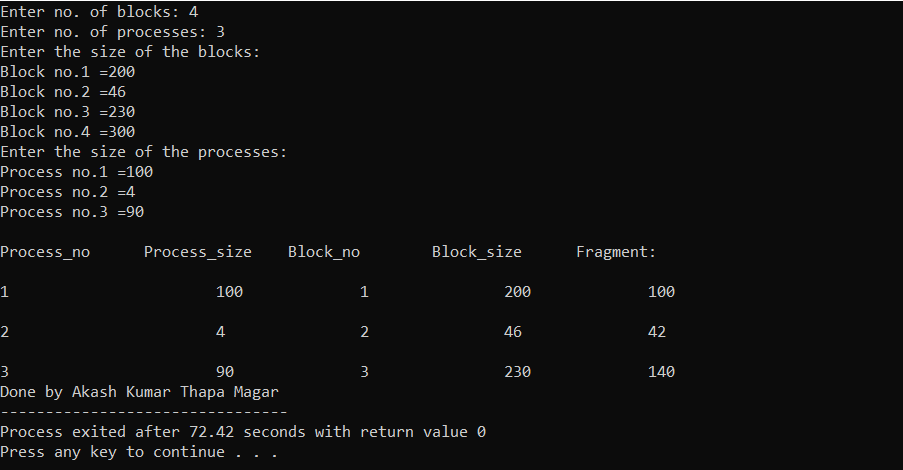
cout<<"\n"<<i<<"\t\t\t"<<p[i]<<"\t\t"<<parray[i]<<"\t\t"<<b[parray[i]]<<"\t\t"<<fragment[i]<<endl;

}

cout<<"Done by Akash Kumar Thapa Magar";

}

**OUTPUT:**



**C program for the producer-consumer problem**

#include <stdio.h>

#include <stdlib.h>

// Initialize a mutex to 1

int mutex = 1;

// Number of full slots as 0

int full = 0;

// Number of empty slots as size

// of buffer

int empty = 10, x = 0;

// Function to produce an item and

// add it to the buffer

void producer()

{

// Decrease mutex value by 1

--mutex;

// Increase the number of full

// slots by 1

++full;

// Decrease the number of empty

// slots by 1

--empty;

// Item produced

x++;

printf("\nProducer produces"

"item %d",

x);

// Increase mutex value by 1

++mutex;

}

// Function to consume an item and

// remove it from buffer

void consumer()

{

// Decrease mutex value by 1

--mutex;

// Decrease the number of full

// slots by 1

--full;

// Increase the number of empty

// slots by 1

++empty;

printf("\nConsumer consumes "

"item %d",

x);

x--;

// Increase mutex value by 1

++mutex;

}

// Driver Code

int main()

{

int n, i;

printf("\n1. Press 1 for Producer"

"\n2. Press 2 for Consumer"

"\n3. Press 3 for Exit");

#pragma omp critical

for (i = 1; i > 0; i++)

{

printf("\nEnter your choice:");

scanf("%d", &n);

switch (n)

{

case 1:

// If mutex is 1 and empty

// is non-zero, then it is

// possible to produce

if ((mutex == 1) && (empty != 0))

{

producer();

}

// Otherwise, print buffer

// is full

else

{

printf("Buffer is full!");

}

break;

case 2:

// If mutex is 1 and full

// is non-zero, then it is

// possible to consume

if ((mutex == 1) && (full != 0))

{

consumer();

}

// Otherwise, print Buffer

// is empty

else

{

printf("Buffer is empty!");

}

break;

case 3:

exit(0);

break;

}

}

Printf(“Done by Akash Kumar Thapa Magar”);

}

**OUTPUT:**



**WAP to simulate word fit**

#include <stdio.h>

#define max 25

int main()

{

int frag[max], b[max], f[max], i, j, nb, nf, temp, highest = 0;

static int bf[max], ff[max];

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) // 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;

}

ff[i] = j;

highest = temp;

}

printf("\nFile\_no:\tFile\_size :\tBlock\_no:\tBlock\_size:\tFragement");

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

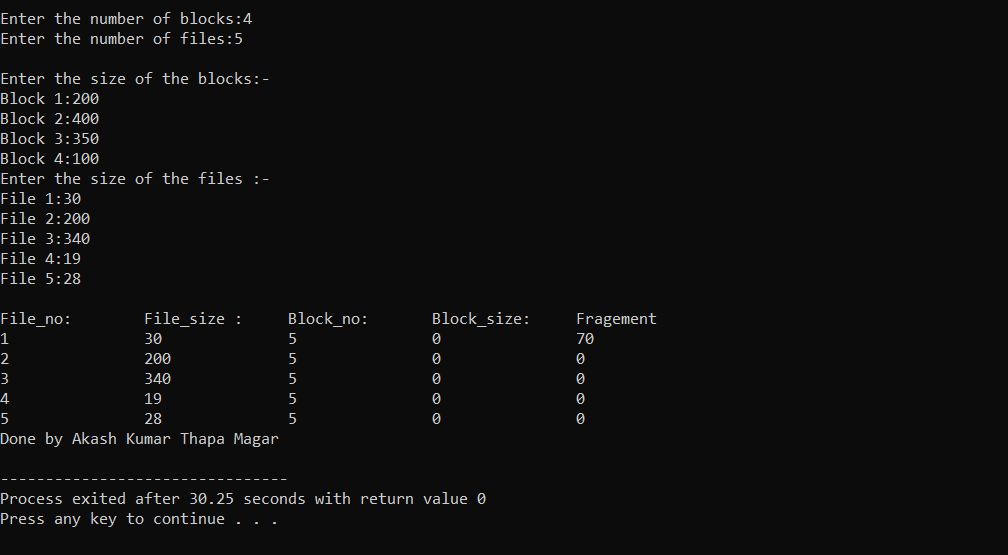
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);

printf("\nDone by Akash Kumar Thapa Magar\n");

return 0;

}

**OUTPUT:**



**WAP to Implement Dining philosopher**

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<semaphore.h>

#include<unistd.h>

sem\_t room;

sem\_t chopstick[5];

void \* philosopher(void \*);

void eat(int);

int main()

{

int i,a[5];

pthread\_t tid[5];

sem\_init(&room,0,4);

for(i=0;i<5;i++)

sem\_init(&chopstick[i],0,1);

for(i=0;i<5;i++){

a[i]=i;

pthread\_create(&tid[i],NULL,philosopher,(void \*)&a[i]);

}

for(i=0;i<5;i++)

pthread\_join(tid[i],NULL);

printf("Done by AKash Kumar Thapa Magar")

}

void \* philosopher(void \* num)

{

int phil=\*(int \*)num;

sem\_wait(&room);

printf("\nPhilosopher %d has entered room",phil);

sem\_wait(&chopstick[phil]);

sem\_wait(&chopstick[(phil+1)%5]);

eat(phil);

sleep(2);

printf("\nPhilosopher %d has finished eating",phil);

sem\_post(&chopstick[(phil+1)%5]);

sem\_post(&chopstick[phil]);

sem\_post(&room);

}

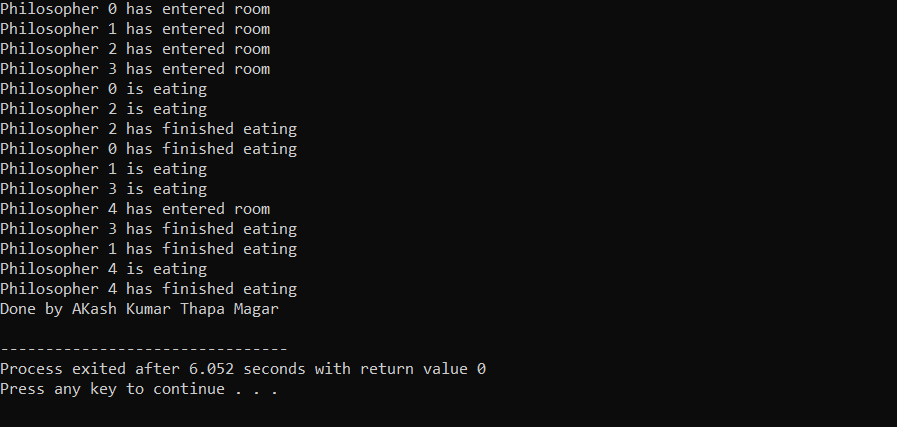
void eat(int phil)

{

printf("\nPhilosopher %d is eating",phil);

}

**OUTPUT:**



**WAP to implement look algorithm in c**

#include<stdio.h>

#include<stdlib.h>

int main()

{

int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;

printf("Enter the number of Requests\n");

scanf("%d",&n);

printf("Enter the Requests sequence\n");

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

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

printf("Enter initial head position\n");

scanf("%d",&initial);

printf("Enter total disk size\n");

scanf("%d",&size);

printf("Enter the head movement direction for high 1 and for low 0\n");

scanf("%d",&move);

// logic for look disk scheduling

/\*logic for sort the request array \*/

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

{

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

{

if(RQ[j]>RQ[j+1])

{

int temp;

temp=RQ[j];

RQ[j]=RQ[j+1];

RQ[j+1]=temp;

}

}

}

int index;

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

{

if(initial<RQ[i])

{

index=i;

break;

}

}

// if movement is towards high value

if(move==1)

{

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

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

initial=RQ[i];

}

for(i=index-1;i>=0;i--)

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

initial=RQ[i];

}

}

// if movement is towards low value

else

{

for(i=index-1;i>=0;i--)

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

initial=RQ[i];

}

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

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

printf("Total head movement is %d \n",TotalHeadMoment);

printf("Done by Akash Kumar Thapa Magar");

return 0;

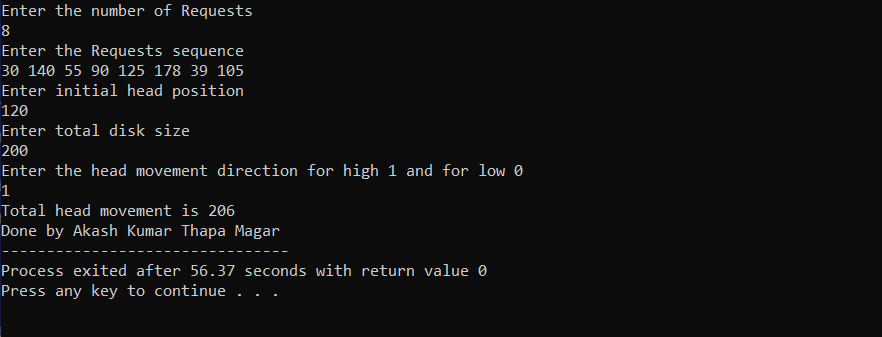
}

initial=RQ[i];

}

}

**OUTPUT:**



**WAP to implement c-look algorithm in c**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int RQ[100], i, j, n, TotalHeadMoment = 0, initial, size, move;

printf("Enter the number of Requests\n");

scanf("%d", &n);

printf("Enter the Requests sequence\n");

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

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

printf("Enter initial head position\n");

scanf("%d", &initial);

printf("Enter total disk size\n");

scanf("%d", &size);

printf("Enter the head movement direction for high 1 and for low 0\n");

scanf("%d", &move);

// logic for C-look disk scheduling

/\*logic for sort the request array \*/

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

{

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

{

if (RQ[j] > RQ[j + 1])

{

int temp;

temp = RQ[j];

RQ[j] = RQ[j + 1];

RQ[j + 1] = temp;

}

}

}

int index;

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

{

if (initial < RQ[i])

{

index = i;

break;

}

}if (move == 1)

{

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

{

TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial); initial =

RQ[i];

}

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

{

TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);

initial = RQ[i];

}

}

else

{

for (i = index - 1; i >= 0; i--)

{

TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial); initial =

RQ[i];

}for (i = n - 1; i >= index; i--)

{TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);

initial = RQ[i];}}

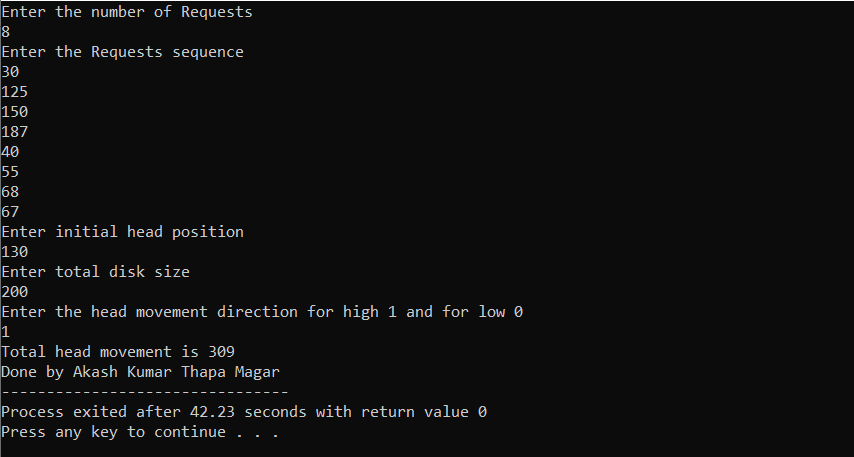
printf("Total head movement is %d \n", TotalHeadMoment);

printf("Done by Akash Kumar Thapa Magar");

return 0;

}

**OUTPUT:**



**Implementing Contiguous file allocation technique**

#include<stdio.h>

#include<conio.h>

#include<windows.h>

int main()

{

int f[50], i, st, len, j, c, k, count = 0;

for(i=0;i<50;i++)

f[i]=0;

printf("\nDone by Akash Kumar Thapa Magar\n");

printf("Files Allocated are : \n");

x: count=0;

printf("Enter starting block and length of files:");

scanf("%d%d", &st,&len);

for(k=st;k<(st+len);k++)

if(f[k]==0)

count++;

if(len==count)

{

for(j=st;j<(st+len);j++)

if(f[j]==0)

{

f[j]=1;

printf("%d\t%d\n",j,f[j]);

}

if(j!=(st+len-1))

printf(" The file is allocated to disk\n");

}

else

printf(" The file is not allocated \n");

printf("Do you want to enter more file(Yes - 1/No - 0)");

scanf("%d", &c);

if(c==1)goto x;

else

exit(0);

return 0;

}

**OUTPUT:**

