```
//Fcfs

#include<stdio.h>
struct pro{
    int pid,at,bt,ct,tat,wt;
};

void read(struct pro a[],int n){
    for(int i=0;i<n;i++){
        printf("Enter the process id: ");
        scanf("%d",&a[i].pid);
        printf("Enter the arrival time: ");
        scanf("%d",&a[i].at);
        printf("Enter the buffer time: ");
        scanf("%d",&a[i].bt);
    }
}

void sort(struct pro a[],int n){
    struct pro temp;
    for(int i=0;i<n-1;i++){
        for(int j=0;j<n-i-1;j++){
            if(a[j].at>a[j+1].at){
                temp=a[j+1];
                a[j+1]=a[j];
                a[j]=temp;
            }
        }
    }
}

void calc(struct pro a[],int n,float *tat,float *wt){
    a[0].ct=a[0].bt+a[0].at;
    a[0].tat=a[0].ct-a[0].at;
    a[0].wt=a[0].tat-a[0].bt;
    for(int i=1;i<n;i++){
        a[i].ct=a[i-1].ct+a[i].bt;
        a[i].tat=a[i].ct-a[i].at;
        a[i].wt=a[i].tat-a[i].bt;
    }
    int tat_s=0,wt_sum=0;

    for(int i=0;i<n;i++){
        tat_s+=a[i].tat;
        wt_sum+=a[i].wt;
    }
    *tat= (float) tat_s/n;
    *wt =  (float) wt_sum/n;
}
```

```
void print(struct pro a[],int n,int c){
   if(c){
    printf("\nafter scheduling:\npid\tat\tbt\tct\ttat\twt\n----
----------------------------------------\n");
    for(int i=0;i<n;i++){

printf("%d\t%d\t%d\t%d\t%d\t%d\n",a[i].pid,a[i].at,a[i].bt,a[i]
.ct,a[i].tat,a[i].wt);
    }
    printf("\n");
    }
    else{
        printf("\nbefore scheduling:\npid\tat\tbt\n------------
-------------------\n");
    for(int i=0;i<n;i++){
        printf("%d\t%d\t%d\n",a[i].pid,a[i].at,a[i].bt);
    }
    printf("\n");
    }
}

void main(){
    int a;
    float avg_tat=0,avg_wt=0;
    printf("Enter the no of processes: ");
    scanf("%d",&a);
    struct pro ar[a];
    read(ar,a);
    print(ar,a,0);
    sort(ar,a);
    calc(ar,a,&avg_tat,&avg_wt);
    print(ar,a,1);
    printf("average waiting time: %.2fms",avg_wt);
    printf("\naverage turn around time: %.2fms",avg_tat);
}
```

//Fcfs output:

```
Enter the no of processes: 5
Enter the process id: 0
Enter the arrival time: 0
Enter the buffer time: 2
Enter the process id: 1
Enter the arrival time: 1
Enter the buffer time: 6
Enter the process id: 2
Enter the arrival time: 2
Enter the buffer time: 4
Enter the process id: 3
Enter the arrival time: 3
Enter the buffer time: 9
Enter the process id: 4
Enter the arrival time: 6
Enter the buffer time: 2

before scheduling:
pid     at      bt
-------------------------------
0       0       2
1       1       6
2       2       4
3       3       9
4       6       2


after scheduling:
pid     at      bt      ct      tat     wt
-----------------------------------------------
0       0       2       2       2       0
1       1       6       8       7       1
2       2       4       12      10      6
3       3       9       21      18      9
4       6       2       23      17      15

average waiting time: 6.20ms
average turn around time: 10.80ms
```

```
//Sjf
#include<stdio.h>

struct process{
	int pid,at,bt,ct,wt,tat;
};

void read(struct process ar[],int n){
	for(int i=0;i<n;i++){
		printf("Enter the arrival time of p%d: ",i+1);
		scanf("%d",&ar[i].at);
		printf("Enter the burst time of p%d: ",i+1);
		scanf("%d",&ar[i].bt);
		ar[i].pid = i+1;
	}
}
void sort(struct process ar[],int n){
	struct process temp;
	for(int i=0;i<n-1;i++){
		for(int j=0;j<n-1-i;j++){
			if(ar[j].at>ar[j+1].at){
				temp=ar[j];
				ar[j]=ar[j+1];
				ar[j+1]=temp;
			}
		}
	}
}

void work(struct process ar[],int n){
	int temp,small;
	struct process Temp;
	ar[0].ct=ar[0].at + ar[0].bt;
	ar[0].tat = ar[0].ct - ar[0].at;
	ar[0].wt = ar[0].tat - ar[0].bt;
	temp=ar[0].ct;
	for(int i=1;i<n;i++){
		small=i;
		for(int j=i;j<n;j++){
			if(ar[j].bt<ar[small].bt && ar[j].at<=temp){
				small = j;
			}
		}
		Temp= ar[i];
		ar[i]=ar[small];
		ar[small]=Temp;
	ar[i].ct=ar[i-1].ct + ar[i].bt;
	  ar[i].tat = ar[i].ct - ar[i].at;
	  ar[i].wt = ar[i].tat - ar[i].bt;
```

```
      temp=ar[i].ct;
      }
}

void print(struct process ar[],int n){
    printf("\nafter scheduling:\npid\tat\tbt\tct\ttat\twt\n--
----------------------------------------\n");
    for(int i=0;i<n;i++){

printf("%d\t%d\t%d\t%d\t%d\t%d\n",ar[i].pid,ar[i].at,ar[i].bt
,ar[i].ct,ar[i].tat,ar[i].wt);
    }
    printf("\n");
}

void calc(struct process ar[],int n,float *t_wt,float
*t_tat){
      int total_wt=0,total_tat=0;
      for (int i = 0; i < n; ++i)
      {
            total_wt+=ar[i].wt;
            total_tat+=ar[i].tat;
      }
      *t_tat=total_tat;
      *t_wt=total_wt;
}
void main(){
      int n;
      float t_wt,t_tat;
      printf("Enter the no of processes: ");
      scanf("%d",&n);
      struct process ar[n];
      read(ar,n);
      sort(ar,n);
      work(ar,n);
      print(ar,n);
      calc(ar,n,&t_wt,&t_tat);
      printf("\naverage waiting time= %0.2fms\naverage turn
around time=%0.2fms\n",t_wt/n,t_tat/n);
}
```

```
//Sjf Output

Enter the no of processes: 5
Enter the arrival time of p1: 1
Enter the burst time of p1: 7
Enter the arrival time of p2: 3
Enter the burst time of p2: 3
Enter the arrival time of p3: 6
Enter the burst time of p3: 2
Enter the arrival time of p4: 7
Enter the burst time of p4: 10
Enter the arrival time of p5: 9
Enter the burst time of p5: 8

after scheduling:
-----------------------------------------
1       1       7       8       7       0
3       6       2       10      4       2
2       3       3       13      10      7
5       9       8       21      12      4
4       7       10      31      24      14


average waiting time= 5.40ms
average turn around time=11.40ms
```

```
//Round robin
#include<stdio.h>
struct sjf
{
    int pno;
    int at;
    int bt;
    int wt;
    int ct;
    int tat;
    int rebt;
};
void sort(struct sjf s[],int n);
void display(struct sjf s[],int n);
void main()
{
    struct sjf s[10];
    int rq[20],k,ts,l;
    int n,i,j,flag[10];
    float avg_wt=0,avg_tat=0;
    printf("Enter the number of processes:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter the processes no:");
        scanf("%d",&s[i].pno);
        printf("Enter the arrival time of processes
%d:",s[i].pno);
        scanf("%d",&s[i].at);
        printf("Enter the burst time of processes %d:",s[i].pno);
        scanf("%d",&s[i].bt);
        s[i].wt=0;
        s[i].ct=0;
        s[i].tat=0;
        flag[i]=0;
        s[i].rebt=s[i].bt;
    }
    printf("Enter time slice:");
    scanf("%d",&ts);
    printf("Before Scheduling\n");
    printf("*****************\n");
    display(s,n);
    sort(s,n);
    display(s,n);
     // For calculating the completion time.
    //first process
    if(s[0].bt<ts)
    {
        s[0].ct+=s[0].bt+s[0].at;
        s[0].rebt=0;
```

```
}
else
{
    s[0].ct+=ts+s[0].at;
    s[0].rebt=s[0].rebt-ts;
}
flag[0]=1;//taken this process
k=1;
for(i=1;i<n;i++)
{
    if(s[i].at<=s[0].ct)
    {
        rq[k++]=i;
        flag[i]=1;
    }
}
if(s[0].rebt>0)
    rq[k++]=0;
for(i=1;i<k;i++)//take process from ready queue
{
    if(s[rq[i]].rebt<ts)
    {

        s[rq[i]].ct=s[rq[i-1]].ct+s[rq[i]].rebt;
        s[rq[i]].rebt=0;
    }
    else
    {
        s[rq[i]].ct=s[rq[i-1]].ct+ts;
        s[rq[i]].rebt=s[rq[i]].rebt-ts;
    }
    for(j=0;j<n;j++)
    {
        if(s[j].at<=s[rq[i]].ct && flag[j]==0)
        {
            rq[k++]=j;
            flag[j]=1;
        }
    }
    if(s[rq[i]].rebt>0)
        rq[k++]=rq[i];
}
for(l=0;l<k;l++)
    printf("Ready Queue:%d  ",s[rq[l]].pno);
// For calculating the turn around time.
for(i=0;i<n;i++)
{
    s[i].tat+=s[i].ct-s[i].at;
}
// For calculating the waiting time.
```

```
    for(i=0;i<n;i++)
    {
        s[i].wt=s[i].tat-s[i].bt;
    }
    printf("\nAfter Scheduling\n");
    printf("****************\n");
    display(s,n);
    // For calculating the Avg waiting time.
    for(i=0;i<n;i++)
    {
        avg_wt+=s[i].wt;
    }
    avg_wt=(float)avg_wt/n;
    for(i=0;i<n;i++)
    {
        avg_tat+=s[i].tat;
    }
    avg_tat=avg_tat/(float)n;
    printf("\nAverage waiting time=%0.2fms",avg_wt);
    printf("\nAverage turn around time=%0.2fms\n",avg_tat);
}
void sort(struct sjf s[],int n)
{
    int i,j;
    struct sjf temp;
    for(i=0;i<n;i++)
        for(j=0;j<n-i-1;j++)
        {
            if((s[j].at>s[j+1].at ) )
            {
                temp=s[j];
                s[j]=s[j+1];
                s[j+1]=temp;
            }
        }

}
void display(struct sjf s[],int n)
{
    int i;
    printf("Process No    Arr.Time   Burst.Time     Wait.Time
Com.Time    Turn aroud.Time\n");
    for(i=0;i<n;i++)
    {

printf("%d\t\t%d\t\t%d\t%d\t\t%d\t\t%d\n",s[i].pno,s[i].at,s[i].b
t,s[i].wt,s[i].ct,s[i].tat);
    }
}
```

//Round Robin Output

```
Enter the number of processes:5
Enter the processes no:1
Enter the arrival time of processes 1:0
Enter the burst time of processes 1:8
Enter the processes no:2
Enter the arrival time of processes 2:4
Enter the burst time of processes 2:9
Enter the processes no:3
Enter the arrival time of processes 3:3
Enter the burst time of processes 3:2
Enter the processes no:4
Enter the arrival time of processes 4:5
Enter the burst time of processes 4:5
Enter the processes no:5
Enter the arrival time of processes 5:2
Enter the burst time of processes 5:4
Enter time slice:4
Before Scheduling
*****************
Process No    Arr.Time   Burst.Time     Wait.Time    Com.Time    Turn
aroud.Time
1               0              8       0              0              0
2               4              9       0              0              0
3               3              2       0              0              0
4               5              5       0              0              0
5               2              4       0              0              0
Process No    Arr.Time   Burst.Time     Wait.Time    Com.Time    Turn
aroud.Time
1               0              8       0              0              0
5               2              4       0              0              0
3               3              2       0              0              0
2               4              9       0              0              0
4               5              5       0              0              0
Ready Queue:1  Ready Queue:5  Ready Queue:3  Ready Queue:2  Ready Queue:1
Ready Queue:4  Ready Queue:2  Ready Queue:4  Ready Queue:2

After Scheduling
***************
Process No    Arr.Time   Burst.Time     Wait.Time    Com.Time    Turn
aroud.Time
1               0              8       6              14             14
5               2              4       -2             4              2
3               3              2       1              6              3
2               4              9       11             24             20
4               5              5       13             23             18

Average waiting time=5.80ms
Average turn around time=11.40ms
```

```
//Priority

#include<stdio.h>
struct process{
      int pid,at,bt,ct,wt,tat,pr;
};
void read(struct process ar[],int n){
      for(int i=0;i<n;i++){
            printf("Enter the arrival time of p%d: ",i+1);
            scanf("%d",&ar[i].at);
            printf("Enter the burst time of p%d: ",i+1);
            scanf("%d",&ar[i].bt);
        printf("enter the priority: ");
        scanf("%d",&ar[i].pr);
            ar[i].pid = i+1;
}}

void sort(struct process ar[],int n){
      struct process temp;
      for(int i=0;i<n-1;i++){
            for(int j=0;j<n-1-i;j++){
                  if(ar[j].at>ar[j+1].at){
                        temp=ar[j];
                        ar[j]=ar[j+1];
                        ar[j+1]=temp;
}}}}
void work(struct process ar[],int n){
      int temp,tag;
      struct process Temp;
      ar[0].ct=ar[0].at + ar[0].bt;
      ar[0].tat = ar[0].ct - ar[0].at;
      ar[0].wt = ar[0].tat - ar[0].bt;
      temp=ar[0].ct;
      for(int i=1;i<n;i++){
            tag=i;
            for(int j=i;j<n;j++){
                  if(ar[j].pr>ar[tag].pr && ar[j].at<=temp){
//use ar[j].pr>ar[tag].pr when higher number represent higher
priority
                        tag = j;
//ar[j].pr<ar[tag].pr  when lower number represent higher
priority
                  }
            }
            Temp= ar[i];
            ar[i]=ar[tag];
            ar[tag]=Temp;
      ar[i].ct=ar[i-1].ct + ar[i].bt;
        ar[i].tat = ar[i].ct - ar[i].at;
        ar[i].wt = ar[i].tat - ar[i].bt;
```

```
      temp=ar[i].ct;
}}

void print_after(struct process ar[],int n){
    printf("\nafter scheduling:\npid\tat\tbt\tct\ttat\twt\n------
-------------------------------------\n");
    for(int i=0;i<n;i++){

printf("%d\t%d\t%d\t%d\t%d\t%d\n",ar[i].pid,ar[i].at,ar[i].bt,ar[
i].ct,ar[i].tat,ar[i].wt);
    }
    printf("\n");
}
void print_before(struct process ar[],int n){
    printf("\nbefore scheduling:\npid\tat\tbt\n-----------------
--\n");
    for(int i=0;i<n;i++){
        printf("%d\t%d\t%d\n",ar[i].pid,ar[i].at,ar[i].bt);    }
    printf("\n");
}

void calc(struct process ar[],int n,float *avg_tat,float
*avg_wt){
      float tat=0,wt=0;
      for (int i = 0; i < n; i++){
            tat+=ar[i].tat;
            wt+=ar[i].wt;
      }
      *avg_tat=tat;
      *avg_wt=wt;
}

void main(){
      int n;
      float total_tat,total_wt;
      printf("Enter the no of processes: ");
      scanf("%d",&n);
      struct process ar[n];
      read(ar,n);
      print_before(ar,n);
      sort(ar,n);
      work(ar,n);
      calc(ar,n,&total_tat,&total_wt);
      print_after(ar,n);
      printf("average waiting time= %.2fms\naverage turn around
time =%.2fms\n",total_wt/n,total_tat/n);
}
```

```
//Priority Output

Enter the no of processes: 5
Enter the arrival time of p1: 0
Enter the burst time of p1: 4
enter the priority: 2
Enter the arrival time of p2: 1
Enter the burst time of p2: 3
enter the priority: 3
Enter the arrival time of p3: 2
Enter the burst time of p3: 1
enter the priority: 4
Enter the arrival time of p4: 3
Enter the burst time of p4: 5
enter the priority: 5
Enter the arrival time of p5: 4
Enter the burst time of p5: 2
enter the priority: 5

before scheduling:
pid     at      bt
--------------------
1       0       4
2       1       3
3       2       1
4       3       5
5       4       2


after scheduling:
pid     at      bt      ct      tat     wt
-------------------------------------------
1       0       4       4       4       0
4       3       5       9       6       1
5       4       2       11      7       5
3       2       1       12      10      9
2       1       3       15      14      11

average waiting time= 5.20ms
average turn around time =8.20ms
```

```
// first-fit – memory management

#include<stdio.h>
struct process{
    int size;
    int allocated;
};
void main(){
    int nb,np;
    printf("enter the no of memory blocks: ");
    scanf("%d",&nb);
    int block[nb];
    printf("enter the size of each memory blocks: ");
    for (int i = 0; i < nb; i++)
    {
        scanf("%d",&block[i]);
    }
    printf("enter the no of processes: ");
    scanf("%d",&np);
    struct process pro[np];
    printf("enter the size of each process: ");
    for (int i = 0; i < np; i++)
    {
        scanf("%d",&pro[i].size);
        pro[i].allocated=-1;
    }
    //code to perform first fit
    for (int  i = 0; i < np; i++)
    {
        for (int  j = 0; j < nb; j++){
            if(pro[i].size<=block[j]){
                pro[i].allocated=block[j];
                block[j]=block[j]-pro[i].size;
                break;
            }
        }
    }
    //code for printing
    printf("\nsi.no\tprocess\t\tblock size");
    for (int i = 0; i < np; i++)
    {
        if (pro[i].allocated!=-1){

printf("\n%d\t%d\t\t%d",i+1,pro[i].size,pro[i].allocated);
        }
        else{
            printf("\n%d\t%d\t\tnot allocatted",i+1,pro[i].size);
        }
    }
}
```

```
//first-fit output

enter the no of memory blocks: 6
enter the size of each memory blocks: 200
450
500
600
300
250
enter the no of processes: 4
enter the size of each process: 347
190
468
475

si.no   process         block size
1       347             450
2       190             200
3       468             500
4       475             600
```

```
//best-fit – memory management

#include<stdio.h>
struct process{
    int size;
    int allocated;
};

void main(){
    int nb,np,best_node;
    printf("enter the no of memory blocks: ");
    scanf("%d",&nb);
    int block[nb];
    printf("enter the size of each memory blocks: ");
    for (int i = 0; i < nb; i++)
    {
        scanf("%d",&block[i]);
    }

    printf("enter the no of processes: ");
    scanf("%d",&np);
    struct process pro[np];
    printf("enter the size of each process: ");
    for (int i = 0; i < np; i++)
    {
        scanf("%d",&pro[i].size);
        pro[i].allocated=-1;
    }

    //code to perform best fit
  for (int  i = 0; i < np; i++)
  {
    best_node=-1;
    for (int j = 0; j < nb; j++)
    {
        if(pro[i].size<=block[j]){
            if(best_node==-1)
                best_node=j;
            else{
                if(block[j]<block[best_node]){
                    best_node=j;
                }
            }
        }
    }
    if(best_node!=-1){
        pro[i].allocated=block[best_node];
        block[best_node]=block[best_node]-pro[i].size;
        }
  }
```

```
    //code for printing
    printf("\nsi.no\tprocess\t\tblock size");
    for (int i = 0; i < np; i++)
    {
        if (pro[i].allocated!=-1)
        {

printf("\n%d\t%d\t\t%d",i+1,pro[i].size,pro[i].allocated);
        }
        else{
            printf("\n%d\t%d\t\tnot allocatted",i+1,pro[i].size);
        }
    }
}



//best-fit output

enter the no of memory blocks: 6
enter the size of each memory blocks: 200
450
500
600
300
250
enter the no of processes: 4
enter the size of each process: 347
190
468
475

si.no   process         block size
1       347             450
2       190             200
3       468             500
4       475             600
```

```
//worst-fit – memory management

#include<stdio.h>
struct process{
    int size;
    int allocated;
};

void main(){
    int nb,np,worst_node;
    printf("enter the no of memory blocks: ");
    scanf("%d",&nb);
    int block[nb];
    printf("enter the size of each memory blocks: ");
    for (int i = 0; i < nb; i++)
    {
        scanf("%d",&block[i]);
    }
    printf("enter the no of processes: ");
    scanf("%d",&np);
    struct process pro[np];
    printf("enter the size of each process: ");
    for (int i = 0; i < np; i++)
    {
        scanf("%d",&pro[i].size);
        pro[i].allocated=-1;
    }

    //code to perform worst fit
  for (int  i = 0; i < np; i++)
  {
    worst_node=-1;
    for (int j = 0; j < nb; j++)
    {
        if(pro[i].size<=block[j]){
            if(worst_node==-1)
                worst_node=j;
            else{
                if(block[j]>block[worst_node]){
                    worst_node=j;
                }
            }
        }
    }
    if(worst_node!=-1){
        pro[i].allocated=block[worst_node];
        block[worst_node]=block[worst_node]-pro[i].size;
        }

  }
```

```
    //code for printing
    printf("\nsi.no\tprocess\t\tblock size");
    for (int i = 0; i < np; i++)
    {
        if (pro[i].allocated!=-1)
        {

printf("\n%d\t%d\t\t%d",i+1,pro[i].size,pro[i].allocated);
        }
        else{
            printf("\n%d\t%d\t\tnot allocatted",i+1,pro[i].size);
        }
    }
}
```

```
//worst-fit output
enter the size of each memory blocks: 200
450
500
600
300
250
enter the no of processes: 4
enter the size of each process: 347
190
468
475

si.no   process         block size
1       347             600
2       190             500
3       468             not allocatted
4       475             not allocated
```

```
//disk scheduling -fcfs

#include<stdio.h>
#include<stdlib.h>
void main(){
    int n,initial,hm=0;
    printf("enter the no of requests: ");
    scanf("%d",&n);
    int ar[n];
    printf("enter the request one by one:\n");
    for (int i = 0; i < n; i++)
    {
        scanf("%d",&ar[i]);
    }
    printf("enter the initial position of the head: ");
    scanf("%d",&initial);
    for (int i = 0; i < n; i++)
    {
        hm=hm+abs(initial-ar[i]);
        initial=ar[i];
    }
    printf("total head movement = %d\n",hm);
}



//disk-fcfs output

enter the no of requests: 8
enter the request one by one:
98
183
37
122
14
124
65
67
enter the initial position of the head: 53
total head movement = 640
```

```
//disk scheduling -scan

#include<stdio.h>
#include<stdlib.h>

void main(){
    int j,i,n,temp,max,initial,headmovement=0,drection,index;
    printf("enter the no of disk requests: ");
    scanf("%d",&n);
    int ar[n];
    printf("enter all the request: ");
    for (int i = 0; i < n; i++)
    {
        scanf("%d",&ar[i]);
    }
    printf("enter the max size of the disk: ");
    scanf("%d",&max);
    max--;
    printf("enter the initial position of the head: ");
    scanf("%d",&initial);
    printf("enter the direction of movement of the head \n 0-low
\n 1-high\n:");
    scanf("%d",&drection);


    //bubble sort
    for (int i = 0; i <n; i++)
    {
        for (int j = 0; j < n-i-1; j++)
        {
            if (ar[j]>ar[j+1])
            {
                temp=ar[j];
                ar[j]=ar[j+1];
                ar[j+1]=temp;
            }

        }

    }

    //finding the index just right to the intial position of the
head
    for (int i = 0; i < n; i++)
    {
        if (ar[i]>initial)
        {
            index=i;break;
        }
        index=i;
```

```
    }


    switch (drection)
    {
    case 1:
        for (i = index; i < n; i++)
        {
            headmovement+=abs(ar[i]-initial);
            initial=ar[i];
        }
        headmovement+=max-initial;
        initial=max;
        for ( i = index-1; i >= 0; i--)
        {
            headmovement+=initial-ar[i];
            initial=ar[i];
        }
        break;

    case 0:
        for (i = index-1; i >= 0; i--)
        {
            headmovement+=abs(initial-ar[i]);
            initial=ar[i];
        }
        headmovement+=initial;
        initial=0;
        for ( i =index; i < n; i++)
        {
            headmovement+=ar[i]-initial;
            initial=ar[i];
        }
        break;

    default:
        exit(0);
    }

    printf("total head movement=%d\n",headmovement);
}
```

//disk-scan output1 – moving towards lower disk address

```
enter the no of disk requests: 8
enter all the request: 176
79
34
60
92
11
41
114
enter the max size of the disk: 200
enter the initial position of the head: 50
enter the direction of movement of the head
 0-low
 1-high
:0
total head movement=226
```

//disk-scan output2 – moving towards higher disk address

```
enter the no of disk requests: 8
enter all the request: 98
183
41
122
14
124
65
67
enter the max size of the disk: 200
enter the initial position of the head: 53
enter the direction of movement of the head
 0-low
 1-high
:1
total head movement=331
```

```
//disk scheduling -cscan

#include<stdio.h>
#include<stdlib.h>

void main(){
    int n,initial,dir,index,headmovement=0,max;
    printf("enter the no of requests: ");
    scanf("%d",&n);
    int ar[n];
    printf("enter the request sequence:\n ");
    for (int i = 0; i < n; i++)
    {
        scanf("%d",&ar[i]);
    }

    int temp;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n-i-1; j++)
        {
            if (ar[j]>ar[j+1])
            {
                temp=ar[j+1];
                ar[j+1]=ar[j];
                ar[j]=temp;
            }
        }
    }

    printf("enter the initial position of the head: ");
    scanf("%d",&initial);
    printf("enter the max no of cylinders: ");
    scanf("%d",&max);
    max--;
    printf("enter the initial head movement:\n 1-high \n 0-low\n:
");
    scanf("%d",&dir);

    for (int i = 0; i < n; i++)
    {
        if (ar[i]>initial)
        {
            index=i;
            break;
        }
        index=i;
    }
```

```
    switch (dir)
    {
    case 1 :
        for (int i = index; i < n; i++)
        {
            headmovement+=abs(initial-ar[i]);
            initial=ar[i];
        }
            headmovement+=max-initial;
            headmovement+=max;
            headmovement+=ar[index-1];
        break;

    case 0 :
    for (int i = index-1; i >= 0; i--)
        {
            headmovement+=abs(initial-ar[i]);
            initial=ar[i];
        }
        headmovement+=initial;
        headmovement+=max;
        headmovement+=max-ar[index];
        break;

    default:
        break;
    }
    printf("total headmovement = %d\n",headmovement);
}
```

```
//disk-cscan output1 – moving towards lower disk address

enter the no of requests: 8
enter the request sequence:
 88
137
122
183
14
133
65
78
enter the initial position of the head: 54
enter the max no of cylinders: 200
enter the initial head movement:
 1-high
 0-low
: 0
total headmovement = 387
```

```
//disk-cscan output2 – moving towards higher disk address

enter the no of requests: 8
enter the request sequence:
 176
79
34
60
92
11
41
114
enter the initial position of the head: 50
enter the max no of cylinders: 200
enter the initial head movement:
 1-high
 0-low
: 1
total headmovement = 389
```

```
//bankers

#include<stdio.h>
void main(){
    int np,nr,flag,k=0;
    printf("enter the no of processes: ");
    scanf("%d",&np);
    printf("enter the no of resources: ");
    scanf("%d",&nr);
    int available[nr],work[nr],ss[np],finish[np];
    int max[np][nr],allocation[np][nr],need[np][nr];
    printf("enter the allocation matrix: \n");
    for (int i = 0; i < np; i++)
    {
        for (int j = 0; j < nr; j++)
        {
            scanf("%d",&allocation[i][j]);
        }

    }
    printf("enter the max matrix: \n");
    for (int i = 0; i < np; i++)
    {
        for (int j = 0; j < nr; j++)
        {
            scanf("%d",&max[i][j]);
        }

    }
    printf("enter the available resources: \n");
    for (int i = 0; i < nr; i++)
    {
        scanf("%d",&available[i]);
        work[i]=available[i];
    }
    for (int i = 0; i < np; i++)
    {
        finish[i]=0;
    }
    printf("need matrix: \n");
    for (int i = 0; i < np; i++)
    {
        for (int j = 0; j < nr; j++)
        {
            need[i][j]=max[i][j]-allocation[i][j];
            printf("%d \t",need[i][j]);
        }
        printf("\n");
    }
    printf("\n");
```

```
    for (int i = 0; i < np; i++)
    {
        for (int j = 0; j < np; j++)
        {
            flag=0;
            if (finish[j]==0)
            {
                for (int a = 0; a < nr; a++)
                {
                    if(need[j][a]>work[a]){
                        flag=1;
                        break;
                    }
                }
                if (flag==0)
                {
                    for (int a = 0; a < nr; a++)
                    {
                        work[a]+=allocation[j][a];
                    }
                    finish[j]=1;
                    ss[k]=j;
                    k++;
                }
            }
        }
    }
    flag=0;
    for (int i = 0; i < np; i++){
        if(finish[i]==0){
            flag=1;
            break;
        }
    }
    if (flag==1)
    {
        printf("system is not safe\n");
    }
    else{
        printf("system is safe\nsafe sequence: ");
        for (int i = 0; i < np; i++)
        {
            printf(" P%d ",ss[i]);
            if (i<np-1)
                printf(" -> ");
        }
        printf("\n");
    }
}
```

```
//bankers output

enter the no of processes: 5
enter the no of resources: 3
enter the allocation matrix:
0
1
0
2
0
0
3
0
2
2
1
1
0
0
2
enter the max matrix:
7
5
3
3
2
2
9
0
2
2
2
2
4
3
3
enter the available resources:
3
3
2
need matrix:
7       4       3
1       2       2
6       0       0
0       1       1
4       3       1

system is safe
safe sequence:  P1  ->  P3  ->  P4  ->  P0  ->  P2
```

```
//pass1

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void passOne(char label[10], char opcode[10], char operand[10],
char code[10], char mnemonic[3]);
void display();
int main()
{
    // for reading from input
    char label[10], opcode[10], operand[10];
    // for reading from optab
    char code[10], mnemonic[3];
    // call the function
    passOne(label, opcode, operand, code, mnemonic);
    return 0;
}
void passOne(char label[10], char opcode[10], char operand[10],
char code[10], char mnemonic[3])
{
    int locctr, start, length;
    FILE *fp1, *fp2, *fp3, *fp4, *fp5; // file pointers
    // read mode
    fp1 = fopen("input.txt", "r");
    fp2 = fopen("optab.txt", "r");
    // write mode
    fp3 = fopen("symtab.txt", "w");
    fp4 = fopen("intermediate.txt", "w");
    fp5 = fopen("length.txt", "w");
    fscanf(fp1, "%s\t%s\t%s", label, opcode, operand);// read
first line
    if (strcmp(opcode, "START") == 0){
        // atoi() requires stdlib.h header file , it converts
ASCII to integer
        start = atoi(operand);// convert operand value from
string to integer and assign to start
        locctr = start;
        fprintf(fp4, "\t%s\t%s\t%s\n", label, opcode, operand);//
write to output file (additional tab space as start will not have
any locctr)
        fscanf(fp1, "%s\t%s\t%s", label, opcode, operand);// read
next line
    }
    else
    {
        locctr = 0;
    }
    // iterate till end
    while (strcmp(opcode, "END") != 0)
```

```
    {
        // transfer address and read line to output file
        fprintf(fp4, "%d\t%s\t%s\t%s\n", locctr, label, opcode,
operand);
        // make symtab file with values not starting with **
        if (strcmp(label, "**") != 0)
        {
            fprintf(fp3, "%s\t%d\n", label, locctr);
        }
        // read from optab (code and mnemonic value)
        fscanf(fp2, "%s\t%s", code, mnemonic);
        // traverse till the end of optab file
        while (strcmp(code, "END") != 0)
        {
            if (strcmp(opcode, code) == 0)
            {                          // if opcode in input
matches the one in optab, increment locctr by 3
                locctr += 3;
                break;
            }
            // read next line
            fscanf(fp2, "%s\t%s", code, mnemonic);
        }
        // Searching opcode for WORD, RESW, BYTE, RESB keywords
and updating locctr
        // WORD -> add 3 to locctr
        if (strcmp(opcode, "WORD") == 0)
        {
            locctr += 3;
        }
                // RESW -> add 3*operand to locctr
        else if (strcmp(opcode, "RESW") == 0)
        {
            locctr += (3 * (atoi(operand))); // convert operand
to integer and multiply with 3
        }
                // BYTE -> add 1 to locctr
        else if (strcmp(opcode, "BYTE") == 0)
        {
            locctr++;
        }
                // RESB -> add operand to locctr
        else if (strcmp(opcode, "RESB") == 0)
        {
            locctr += atoi(operand);
        }
            // read next line
        fscanf(fp1, "%s\t%s\t%s", label, opcode, operand);
    }
    //  transfer last line to file
```

```
    fprintf(fp4, "%d\t%s\t%s\t%s\n", locctr, label, opcode,
operand);
    // Close all files
    fclose(fp4);
    fclose(fp3);
    fclose(fp2);
    fclose(fp1);
    // 8. display outputs
    display();
        // calculate length of program
    length = locctr - start;
    fprintf(fp5, "%d", length);
    fclose(fp5);
    printf("\nThe length of the code : %d\n", length);
}
void display()
{
    char str;
    FILE *fp1, *fp2, *fp3;
    // display content of Input Table
    printf("\nThe contents of Input Table :\n\n");
    fp1 = fopen("input.txt", "r");
    str = fgetc(fp1);
    while (str != EOF)
    {
        printf("%c", str);
        str = fgetc(fp1);
    }
    fclose(fp1);
    // display content of  Output Table
    printf("\n\nThe contents of Output Table :\n\n");
    fp2 = fopen("intermediate.txt", "r");
    str = fgetc(fp2);
    while (str != EOF)
    {
        printf("%c", str);
        str = fgetc(fp2);
    }
    fclose(fp2);
    // display content of  Symtable
    printf("\n\nThe contents of Symbol Table :\n\n");
    fp3 = fopen("symtab.txt", "r");
    str = fgetc(fp3);
    while (str != EOF)
    {
        printf("%c", str);
        str = fgetc(fp3);
    }
    fclose(fp3);
}
```

```
//pass1 output

The contents of Input Table :

**      START   2000
**      LDA     FIVE
**      STA     ALPHA
**      LDCH    CHARZ
**      STCH    C1
ALPHA   RESW    2
FIVE    WORD    5
CHARZ   BYTE    C'Z'
C1      RESB    1
**      END     **


The contents of Output Table :

        **      START   2000
2000    **      LDA     FIVE
2003    **      STA     ALPHA
2006    **      LDCH    CHARZ
2009    **      STCH    C1
2012    ALPHA   RESW    2
2018    FIVE    WORD    5
2021    CHARZ   BYTE    C'Z'
2022    C1      RESB    1
2023    **      END     **


The contents of Symbol Table :

ALPHA   2012
FIVE    2018
CHARZ   2021
C1      2022

The length of the code : 23
```

//input.txt

```
**    START 2000
**    LDA   FIVE
**    STA   ALPHA
**    LDCH  CHARZ
**    STCH  C1
ALPHA       RESW  2
FIVE  WORD  5
CHARZ       BYTE  C'Z'
C1    RESB  1
**    END   **
```

//optab.txt

```
LDA 03
STA   0f
LDCH  53
STCH  57
END   *
```

//symtab.txt

```
ALPHA 2012
FIVE  2018
CHARZ 2021
C1    2022
```

//intermediate.txt

```
      **    START 2000
2000  **    LDA   FIVE
2003  **    STA   ALPHA
2006  **    LDCH  CHARZ
2009  **    STCH  C1
2012  ALPHA RESW  2
2018  FIVE  WORD  5
2021  CHARZ BYTE  C'Z'
2022  C1    RESB  1
2023  **    END   **
```

//length.txt

```
23
```

```
//pass2

/*
!important
Must Compile and Execute 'pass1.c' before executing this
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void display();
// itoa manual implementation as its not ANSI Standard
//start of itoa block
// Function to swap two numbers
void swap(char *x, char *y)
{
    char t = *x; *x = *y; *y = t;
}

// Function to reverse `buffer[i…j]`
char* reverse(char *buffer, int i, int j)
{
    while (i < j)
    {
        swap(&buffer[i++], &buffer[j--]);
    }

    return buffer;
}

// Iterative function to implement `itoa()` function in C
char* itoa(int value, char* buffer, int base)
{
    // invalid input
    if (base < 2 || base > 32)
    {
        return buffer;
    }

    // consider the absolute value of the number
    int n = abs(value);

    int i = 0;
    while (n)
    {
        int r = n % base;

        if (r >= 10)
        {
```

```
            buffer[i++] = 65 + (r - 10);
        }
        else
        {
            buffer[i++] = 48 + r;
        }

        n = n / base;
    }

    // if the number is 0
    if (i == 0)
    {
        buffer[i++] = '0';
    }

    // If the base is 10 and the value is negative, the resulting
string
    // is preceded with a minus sign (-)
    // With any other base, value is always considered unsigned
    if (value < 0 && base == 10)
    {
        buffer[i++] = '-';
    }

    buffer[i] = '\0'; // null terminate string

    // reverse the string and return it
    return reverse(buffer, 0, i - 1);
}
//end of itoa block

int main()
{
    char a[10], ad[10], label[10], opcode[10], operand[10],
symbol[10];
    int start, diff, i, address, add, len, actual_len, finaddr,
prevaddr, j = 0;
    char mnemonic[15][15] = {"LDA", "STA", "LDCH", "STCH"};
    char code[15][15] = {"33", "44", "53", "57"};

    FILE *fp1, *fp2, *fp3, *fp4;
    fp1 = fopen("output.txt", "w");
    fp2 = fopen("symtab.txt", "r");
    fp3 = fopen("intermediate.txt", "r");
    fp4 = fopen("objcode.txt", "w");
    fscanf(fp3, "%s\t%s\t%s", label, opcode, operand);

    while (strcmp(opcode, "END") != 0)
    {
```

```
        prevaddr = address;
        fscanf(fp3, "%d%s%s%s", &address, label, opcode,
operand);
    }
    finaddr = address;
    fclose(fp3);
    fp3 = fopen("intermediate.txt", "r");

    fscanf(fp3, "\t%s\t%s\t%s", label, opcode, operand);
    if (strcmp(opcode, "START") == 0)
    {
        fprintf(fp1, "\t%s\t%s\t%s\n", label, opcode, operand);
        fprintf(fp4, "H^%s^00%s^00%d\n", label, operand,
finaddr);
        fscanf(fp3, "%d%s%s%s", &address, label, opcode,
operand);
        start = address;
        diff = prevaddr - start;
        fprintf(fp4, "T^00%d^%d", address, diff);
    }

    while (strcmp(opcode, "END") != 0)
    {
        if (strcmp(opcode, "BYTE") == 0)
        {
            fprintf(fp1, "%d\t%s\t%s\t%s\t", address, label,
opcode, operand);
            len = strlen(operand);
            actual_len = len - 3;
            fprintf(fp4, "^");
            for (i = 2; i < (actual_len + 2); i++)
            {
                itoa(operand[i], ad, 16);
                fprintf(fp1, "%s", ad);
                fprintf(fp4, "%s", ad);
            }
            fprintf(fp1, "\n");
        }

        else if (strcmp(opcode, "WORD") == 0)
        {
            len = strlen(operand);
            itoa(atoi(operand), a, 10);
            fprintf(fp1, "%d\t%s\t%s\t%s\t00000%s\n", address,
label, opcode, operand, a);
            fprintf(fp4, "^00000%s", a);
        }

        else if ((strcmp(opcode, "RESB") == 0) || (strcmp(opcode,
"RESW") == 0))
```

```
        {
            fprintf(fp1, "%d\t%s\t%s\t%s\n", address, label,
opcode, operand);
        }

        else
        {
            while (strcmp(opcode, mnemonic[j]) != 0)
                j++;
            if (strcmp(operand, "COPY") == 0)
                fprintf(fp1, "%d\t%s\t%s\t%s\t%s0000\n", address,
label, opcode, operand, code[j]);
            else
            {
                rewind(fp2);
                fscanf(fp2, "%s%d", symbol, &add);
                while (strcmp(operand, symbol) != 0)
                    fscanf(fp2, "%s%d", symbol, &add);
                fprintf(fp1, "%d\t%s\t%s\t%s\t%s%d\n", address,
label, opcode, operand, code[j], add);
                fprintf(fp4, "^%s%d", code[j], add);
            }
        }

        fscanf(fp3, "%d%s%s%s", &address, label, opcode,
operand);
    }

    fprintf(fp1, "%d\t%s\t%s\t%s\n", address, label, opcode,
operand);
    fprintf(fp4, "\nE^00%d", start);

    fclose(fp4);
    fclose(fp3);
    fclose(fp2);
    fclose(fp1);

    display();

    return 0;
}

void display()
{
    char ch;
    FILE *fp1, *fp2, *fp3, *fp4;

    printf("\nIntermediate file is converted into object code");

    printf("\n\nThe contents of Intermediate file:\n\n");
```

```
    fp3 = fopen("intermediate.txt", "r");
    ch = fgetc(fp3);
    while (ch != EOF)
    {
        printf("%c", ch);
        ch = fgetc(fp3);
    }
    fclose(fp3);

    printf("\n\nThe contents of Symbol Table :\n\n");
    fp2 = fopen("symtab.txt", "r");
    ch = fgetc(fp2);
    while (ch != EOF)
    {
        printf("%c", ch);
        ch = fgetc(fp2);
    }
    fclose(fp2);

    printf("\n\nThe contents of Output file :\n\n");
    fp1 = fopen("output.txt", "r");
    ch = fgetc(fp1);
    while (ch != EOF)
    {
        printf("%c", ch);
        ch = fgetc(fp1);
    }
    fclose(fp1);

    printf("\n\nThe contents of Object code file :\n\n");
    fp4 = fopen("objcode.txt", "r");
    ch = fgetc(fp4);
    while (ch != EOF)
    {
        printf("%c", ch);
        ch = fgetc(fp4);
    }
    fclose(fp4);
}
```

//pass2 output

Intermediate file is converted into object code

The contents of Intermediate file:

```
        **      START   2000
2000    **      LDA     FIVE
2003    **      STA     ALPHA
2006    **      LDCH    CHARZ
2009    **      STCH    C1
2012    ALPHA   RESW    2
2018    FIVE    WORD    5
2021    CHARZ   BYTE    C'Z'
2022    C1      RESB    1
2023    **      END     **
```

The contents of Symbol Table :

```
ALPHA   2012
FIVE    2018
CHARZ   2021
C1      2022
```

The contents of Output file :

```
        **      START   2000
2000    **      LDA     FIVE    332018
2003    **      STA     ALPHA   442012
2006    **      LDCH    CHARZ   532021
2009    **      STCH    C1      572022
2012    ALPHA   RESW    2
2018    FIVE    WORD    5       000005
2021    CHARZ   BYTE    C'Z'    5A
2022    C1      RESB    1
2023    **      END     **
```

The contents of Object code file :

```
H^**^002000^002023
T^002000^22^332018^442012^532021^572022^000005^5A
E^002000
```

```
//input.txt
**    START 2000
**    LDA   FIVE
**    STA   ALPHA
**    LDCH  CHARZ
**    STCH  C1
ALPHA       RESW  2
FIVE  WORD  5
CHARZ       BYTE  C'Z'
C1    RESB  1
**    END   **
```

```
//optab.txt

LDA 03
STA   0f
LDCH  53
STCH  57
END   *
```

```
//intermediate.txt
      **    START 2000
2000  **    LDA   FIVE
2003  **    STA   ALPHA
2006  **    LDCH  CHARZ
2009  **    STCH  C1
2012  ALPHA RESW  2
2018  FIVE  WORD  5
2021  CHARZ BYTE  C'Z'
2022  C1    RESB  1
2023  **    END   **
```

```
//symtab.txt

ALPHA 2012
FIVE  2018
CHARZ 2021
C1    2022
```

```
//length.txt

23
```

```
//objcode.txt
H^***^002000^002023
T^002000^22^332018^442012^532021^572022^000005^5A
E^002000
```

```
//output.txt
      **    START 2000
2000  **    LDA   FIVE  332018
2003  **    STA   ALPHA 442012
2006  **    LDCH  CHARZ 532021
2009  **    STCH  C1    572022
2012  ALPHA RESW  2
2018  FIVE  WORD  5     000005
2021  CHARZ BYTE  C'Z'  5A
2022  C1    RESB  1
2023  **    END   **
```

```
//absolute loader

#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void main()
{
  FILE *fp;
  int i,addr1,l,j,staddr1;
  char
name[10],line[50],name1[10],addr[10],rec[10],ch,staddr[10];
  printf("Enter the name of the program:" );
  scanf("%s",name);
  fp=fopen("objcode.txt","r");
  fscanf(fp,"%s",line);
  for(i=2,j=0;i<7;i++,j++)
          name1[j]=line[i];
    name1[j]='\0';
  printf("name from obj File. %s\n",name1);
  if(strcmp(name,name1)==0)//verify program name
  {
    do
    {
            fscanf(fp,"%s",line);
            if(line[0]=='T')
            {
                    for(i=2,j=0;i<8;i++,j++)//extract starting
address from object file
                        //which is stored from columns 2 to 7 in
text record and store it in an array staddr and
                        //convert into interger using atoi
function
                            staddr[j]=line[i];
                    staddr[j]='\0';
                    staddr1=atoi(staddr);
                    i=12;//start object code from column 12
                    while(line[i]!='\0')
                    {
                      if(line[i]!='^')
                      {
                        printf("00%d \t %c%c\n",
staddr1,line[i],line[i+1]);// each memory location stores 2
character
                        staddr1++;// increment memory address
                        i=i+2;//move to next character
                      }
                      else i++;// if it is ^
                    }
                }
                else if(line[0]=='E')
```

```
            {
                        printf("\nExecution address: ");
            for (i = 2; i < 8; i++)
                printf("%c", line[i]);
            break;
        }
        }
        while(!feof(fp));
  }
  fclose(fp);
}
```

```
//absolute loader output

Enter the name of the program:FIRST
name from obj File. FIRST
002000   03
002001   20
002002   18
002003   0f
002004   20
002005   12
002006   53
002008   21
002009   57
002010   20
002011   22
002012   00
002013   00
002014   05
002015   5a

Execution address: 002000
```

```
//objcode.txt

H^FIRST^002000^002023
T^002000^22^032018^0f2012^532021^572022^000005^5a
E^002000
```

```
//relocation loader

#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void convert(char h[12]);
char bitmask[12];
char bit[12]={0};
void main()
{
        char
add[6],length[10],input[10],binary[12],relocbit,ch,pn[5];
        int start,inp,len,i,address,opcode,addr,actualadd,tlen;
        FILE *fp1,*fp2;
        printf("\n\n Enter the actual starting address : ");
        scanf("%x",&start);
        fp1=fopen("RInput.txt","r");
        fp2=fopen("ROutput.txt","w");
        fscanf(fp1,"%s",input);
        fprintf(fp2," ----------------------------\n");
        fprintf(fp2," ADDRESS\tCONTENT\n");
        fprintf(fp2," ----------------------------\n");
        while(strcmp(input,"E")!=0)
        {
                if(strcmp(input,"H")==0)
                {
                        fscanf(fp1,"%s",pn);
                        fscanf(fp1,"%x",add);
                        fscanf(fp1,"%x",length);
                        fscanf(fp1,"%s",input);
                }
                if(strcmp(input,"T")==0)
                {
                        fscanf(fp1,"%x",&address);
                        fscanf(fp1,"%x",&tlen);
                        fscanf(fp1,"%s",bitmask);
                        address+=start;
                        convert(bitmask);
                        len=strlen(bit);
                        if(len>=11)
                        len=10;
                        for(i=0;i<len;i++)
                        {
                                fscanf(fp1,"%x",&opcode);
                                fscanf(fp1,"%x",&addr);
                                relocbit=bit[i];
                                if(relocbit=='0')
                                        actualadd=addr;
                                else
                                        actualadd=addr+start;
```

```
                                fprintf(fp2,"\n
%x\t\t%x%x\n",address,opcode,actualadd);
                                address+=3;
                        }
                        fscanf(fp1,"%s",input);
                }
        }
        fprintf(fp2," ----------------------------\n");
        fclose(fp1);
        fclose(fp2);
        printf("\n\n The contents of output
file(ROutput.txt)\n");
        fp2=fopen("ROutput.txt","r");
        ch=fgetc(fp2);
        while(ch!=EOF)
        {
                printf("%c",ch);
                ch=fgetc(fp2);
        }
        fclose(fp2);
}


void convert(char h[12])
{
        int i,l;
        strcpy(bit,"");
        l=strlen(h);
        for(i=0;i<l;i++)
        {
                switch(h[i])
                {
                        case '0':
                            strcat(bit,"0");
                                break;
                        case '1':
                            strcat(bit,"1");
                                break;
                        case '2':
                            strcat(bit,"10");
                                break;
                        case '3':
                            strcat(bit,"11");
                                break;
                        case '4':
                            strcat(bit,"100");
                                break;
                        case '5':
                                  strcat(bit,"101");
                                break;
```

```
                        case '6':
                                  strcat(bit,"110");
                                break;
                        case '7':
                            strcat(bit,"111");
                                break;
                        case '8':
                            strcat(bit,"1000");
                                break;
                        case '9':
                            strcat(bit,"1001");
                                break;
                        case 'A':
                            strcat(bit,"1010");
                                break;
                        case 'B':
                                strcat(bit,"1011");
                                break;
                        case 'C':
                            strcat(bit,"1100");
                                break;
                        case 'D':
                            strcat(bit,"1101");
                                break;
                        case 'E':
                            strcat(bit,"1110");
                                break;
                        case 'F':
                            strcat(bit,"1111");
                                break;
                }
        }
}
```

```
//relocation loader output


Enter the actual starting address : 5000

 The contents of output file(ROutput.txt)
 ----------------------------
 ADDRESS        CONTENT
 ----------------------------

  5000          145033

  5003          486039

  5006          105036

  5009          285030

  500c          305015

  500f          486061

  5012          3c5003

  5015          20502a

  5018          1c5039

  501b          30502d

  7500          1d5036

  7503          486061

  7506          185033

  7509          4c1000

  750c          801000

  750f          601003
 ----------------------------
```

//ROutput.txt

```
 ----------------------------
 ADDRESS    CONTENT
 ----------------------------

  5000            145033

  5003            486039

  5006            105036

  5009            285030

  500c            305015

  500f            486061

  5012            3c5003

  5015            20502a

  5018            1c5039

  501b            30502d

  7500            1d5036

  7503            486061

  7506            185033

  7509            4c1000

  750c            801000

  750f            601003
 ----------------------------
```

//RInput.txt

```
H COPY 000000 00107A
T 000000 1E FFC 14 0033 48 1039 10 0036 28 0030 30 0015 48 1061 3C 0003 20 002A 1C 0039 30 002D
T 002500 15 E00 1D 0036 48 1061 18 0033 4C 1000 80 1000 60 1003
E 000000
```