Ex. No: 11 **PAGING TECHNIQUE OF MEMORY MANAGEMENT**

AIM: To implement the Memory management policy- Paging using C.

ALGORITHM

Step 1: Read all the necessary input from the keyboard.
Step 2: Pages - Logical memory is broken into fixed - sized blocks.
Step 3: Frames – Physical memory is broken into fixed – sized blocks.
Step 4: Calculate the physical address using the following
Physical address = (Frame number * Frame size) + offset
Step 5: Display the physical address.

PROGRAM:

```
/*Memory Allocation with Paging Technique */

#include <stdio.h>

struct PageTable {
   int frameNo;
   int presenceBit;
} pTable[10];

int pMemorySize, lMemorySize, pSize, frames, pages, fTable[20], frameNo;

void getInfo() {
   printf("\n\tMEMORY MANAGEMENT USING PAGING\n");
   printf("\nEnter the Size of Physical memory: ");
   scanf("%d", &pMemorySize);
   printf("Enter the size of Logical memory: ");
   scanf("%d", &lMemorySize);
   printf("Enter the partition size: ");
   scanf("%d", &pSize);
   frames = pMemorySize / pSize;
   pages = lMemorySize / pSize;
   printf("\nThe physical memory is divided into %d no. of frames\n", frames);
   printf("The Logical memory is divided into %d no. of pages\n", pages);
}

void assignFrames() {
   int i;
   for (i = 0; i < pages; i++) {
      pTable[i].frameNo = -1;
      pTable[i].presenceBit = -1;
   }

   for (i = 0; i < frames; i++)
      fTable[i] = 32555;

   for (i = 0; i < pages; i++) {
      printf("\nEnter the Frame number where page %d must be placed: ", i);
```

```
        scanf("%d", &frameNo);
        fTable[frameNo] = i;

        if (pTable[i].presenceBit == -1) {
            pTable[i].frameNo = frameNo;
            pTable[i].presenceBit = 1;
        }
    }

    printf("\nPAGE TABLE\n");
    printf("Page Address FrameNo. Presence Bit\n");

    for (i = 0; i < pages; i++)
        printf("%d\t\t%d\t\t%d\n", i, pTable[i].frameNo, pTable[i].presenceBit);

    printf("\nFRAME TABLE\n");
    printf("FrameAddress PageNo\n");

    for (i = 0; i < frames; i++)
        printf("%d\t\t%d\n", i, fTable[i]);
}

void createPhysicalAddr() {
    int lAddr, pAddr, disp, physAddr, bAddr;

    printf("\nProcess to create the Physical Address\n");
    printf("Enter the Base Address: ");
    scanf("%d", &bAddr);
    printf("Enter the Logical Address: ");
    scanf("%d", &lAddr);

    pAddr = lAddr / pSize;
    disp = lAddr % pSize;

    if (pTable[pAddr].presenceBit == 1)
        physAddr = bAddr + (pTable[pAddr].frameNo * pSize) + disp;

    printf("\nThe Physical Address where the instruction is present: %d\n", physAddr);
}

int main() {
    getInfo();
    assignFrames();
    createPhysicalAddr();
    return 0;
}
```

Output:

```
[it@itlab2sys136 12]$ gcc 12.c && ./a.out
        MEMORY MANAGEMENT USING PAGING

Enter the Size of Physical memory: 16

Enter the size of Logical memory: 8

Enter the partition size: 2

The physical memory is divided into 8 no.of frames

The Logical memory is divided into 4 no.of pages

Enter the Frame number where page 0 must be placed: 5


Enter the Frame number where page 1 must be placed: 6


Enter the Frame number where page 2 must be placed: 7


Enter the Frame number where page 3 must be placed: 2


PAGE TABLE

Page Address FrameNo. Presence Bit

0               5               1
1               6               1
2               7               1
3               2               1



        FRAME TABLE

FrameAddress PageNo

0               32555
1               32555
2               3
3               32555
4               32555
5               0
6               1
7               2



        Process to create the Physical Address


Enter the Base Address: 32555

Enter the Logical Address: 2

The Physical Address where the instruction present: 32567
```

Ex. No: 12(a) FIFO PAGE REPLACEMENT ALGORITHM

AIM: To write a C program to implement FIFO page replacement algorithm

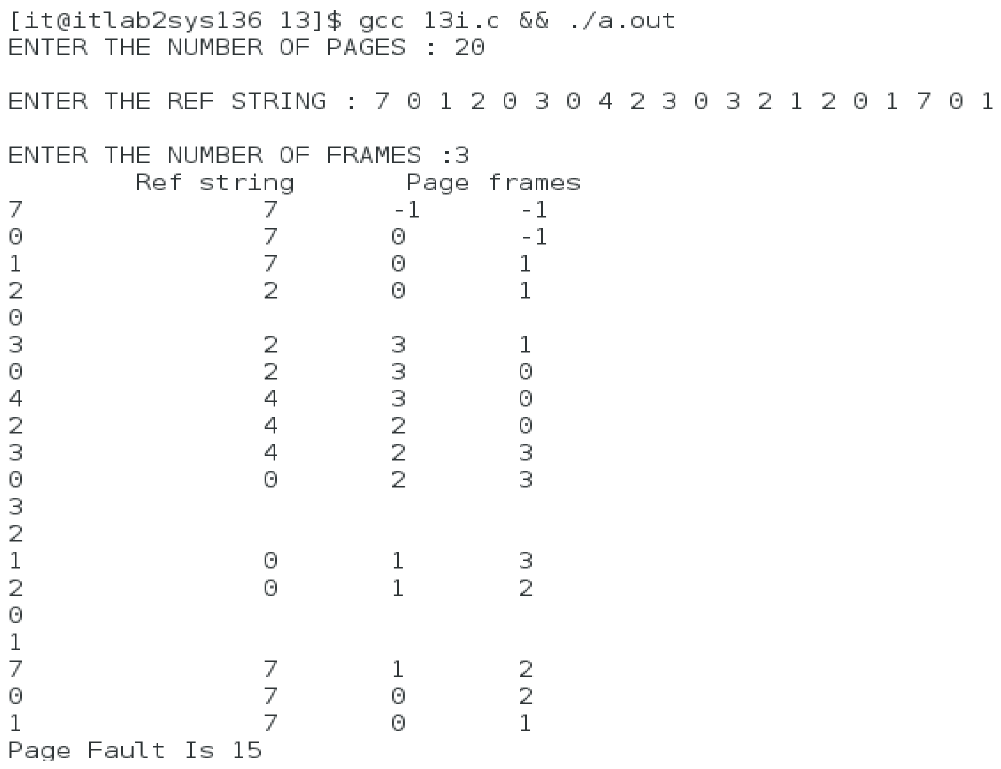
ALGORITHM

1. Declare the size with respect to page length
3. Check the need of replacement from the page to memory
4. Check the need of replacement from old page to new page in memory
5. Form a queue to hold all pages
6. Insert the page require memory into the queue
7. Check for bad replacement and page fault
8. Get the number of processes to be inserted
9. Display the values.

PROGRAM:

```
#include <stdio.h>

int main() {
    int n, a[50], frame[10] = {-1}, no, j = 0, count = 0;

    printf("ENTER THE NUMBER OF PAGES : ");
    scanf("%d", &n);

    printf("\nENTER THE REF STRING : ");
    for (int i = 1; i <= n; i++)
        scanf("%d", &a[i]);

    printf("\nENTER THE NUMBER OF FRAMES :");
    scanf("%d", &no);

    printf("\tRef string\t Page frames\n");

    for (int i = 1; i <= n; i++) {
        printf("%d\t\t", a[i]);
        int avail = 0;

        for (int 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 (int k = 0; k < no; k++)
                printf("%d\t", frame[k]);
        }

        printf("\n");
    }

    printf("Page Fault Is %d\n", count);
    return 0;
}
```

OUTPUT:



```
[it@itlab2sys136 13]$ gcc 13i.c && ./a.out
ENTER THE NUMBER OF PAGES : 20

ENTER THE REF STRING : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

ENTER THE NUMBER OF FRAMES :3
        Ref string       Page frames
7               7       -1      -1
0               7       0       -1
1               7       0       1
2               2       0       1
0
3               2       3       1
0               2       3       0
4               4       3       0
2               4       2       0
3               4       2       3
0               0       2       3
3
2
1               0       1       3
2               0       1       2
0
1
7               7       1       2
0               7       0       2
1               7       0       1
Page Fault Is 15
```

Ex. No: 12(b) **LRU PAGE REPLACEMENT ALGORITHM**

AIM: To implement Least Recently Used Page Replacement algorithm in C.

ALGORITHM

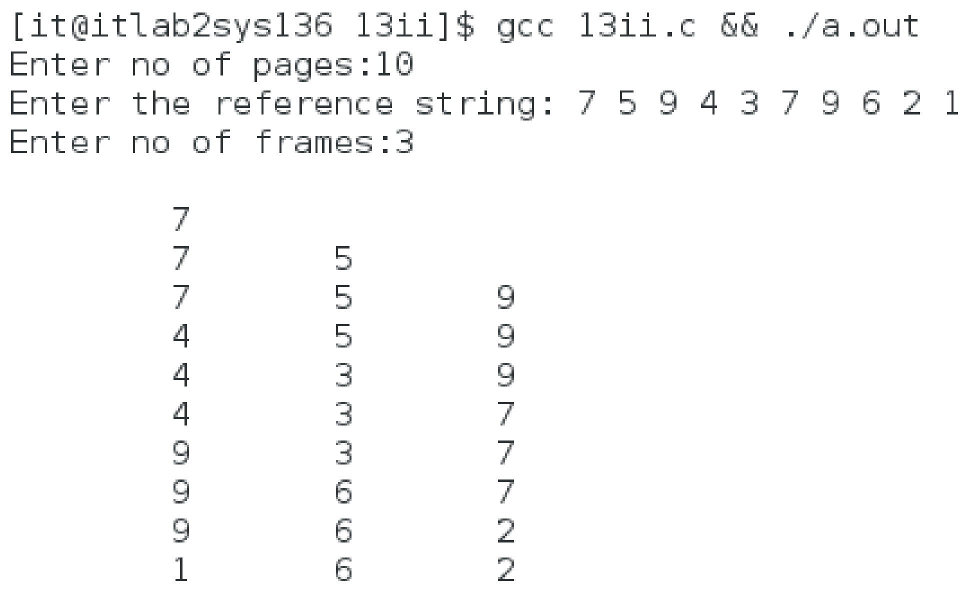
```
Step 1:  Create a queue to hold all pages in memory
Step 2:  When the page is required replace the page at the head of the queue
Step 3:  Now the new page is inserted at the tail of the queue
Step 4:  Create a stack
Step 5:  When the page fault occurs replace page present at the bottom of the stack
```

PROGRAM:

```
#include <stdio.h>

int main() {
   int pageQueue[20], pageReference[50], pageFaults = 0, queueIndex = 0, n, frames, i, j,
recentDist[20], tempDist[20];

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

   printf("Enter the reference string: ");
   for (i = 0; i < n; i++)
      scanf("%d", &pageReference[i]);

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

   pageQueue[queueIndex] = pageReference[0];
   printf("\n\t%d\n", pageQueue[queueIndex++]);

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

      for (j = 0; j < frames; j++) {
         if (pageReference[i] == pageQueue[j])
            notInQueue = 0;
      }

      if (notInQueue) {
         pageFaults++;

         if (queueIndex < frames) {
            pageQueue[queueIndex] = pageReference[i];
            queueIndex++;
```

```
                for (j = 0; j < queueIndex; j++)
                    printf("\t%d", pageQueue[j]);

                printf("\n");
            } else {
                for (j = 0; j < frames; j++) {
                    recentDist[j] = 0;

                    for (int k = i - 1; k < n; k--) {
                        if (pageQueue[j] != pageReference[k])
                            recentDist[j]++;
                        else
                            break;
                    }
                }

                for (j = 0; j < frames; j++)
                    tempDist[j] = recentDist[j];

                for (j = 0; j < frames; j++) {
                    for (int k = j; k < frames; k++) {
                        if (tempDist[j] < tempDist[k]) {
                            int temp = tempDist[j];
                            tempDist[j] = tempDist[k];
                            tempDist[k] = temp;
                        }
                    }
                }

                for (j = 0; j < frames; j++) {
                    if (recentDist[j] == tempDist[0])
                        pageQueue[j] = pageReference[i];
                    printf("\t%d", pageQueue[j]);
                }

                printf("\n");
            }
        }
    }

    printf("\nThe number of page faults is %d\n", pageFaults);

    return 0;
}
```

OUTPUT:



```
[it@itlab2sys136 13ii]$ gcc 13ii.c && ./a.out
Enter no of pages:10
Enter the reference string: 7 5 9 4 3 7 9 6 2 1
Enter no of frames:3

        7
        7       5
        7       5       9
        4       5       9
        4       3       9
        4       3       7
        9       3       7
        9       6       7
        9       6       2
        1       6       2
```

Ex No: 12 (c) LFU PAGE REPLACEMENT ALGORITHM

AIM: To implement LFU page replacement technique.

ALGORITHM

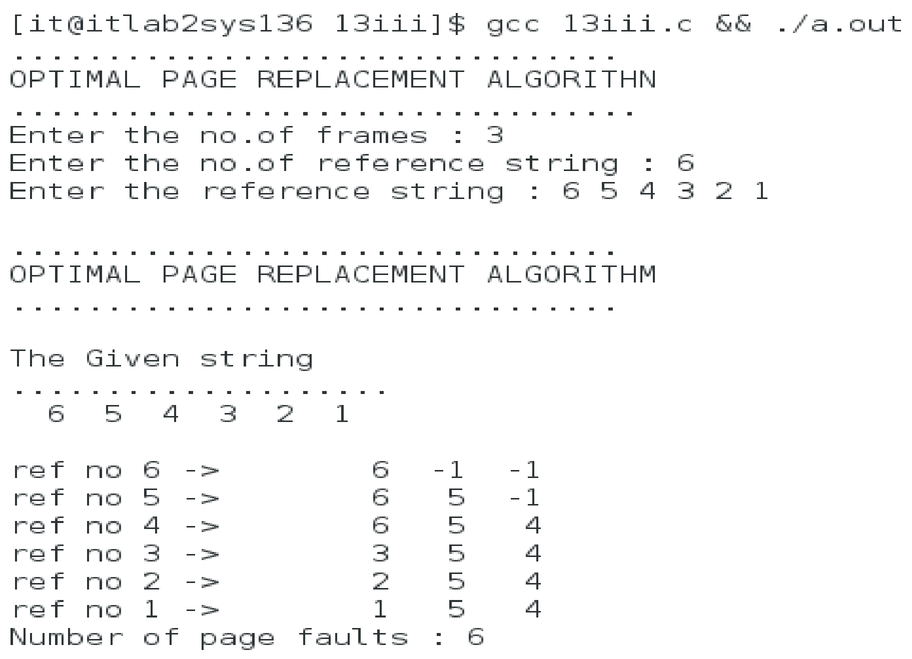
1. Read Number Of Pages And Frames
2. Read Each Page Value
3. Search For Page In The Frames
4. If Not Available Allocate Free Frame
5. If No Frames Is Free Repalce The Page With The Page That Is Least frequently used.
6. Print Page Number Of Page Faults

PROGRAM:

```
#include <stdio.h>

#define MAX_FRAMES 10
#define MAX_REFERENCES 50

int numFrames, numReferences, pageFaults = 0, victim = -1;
int references[MAX_REFERENCES], frames[MAX_FRAMES],
optimalCal[MAX_FRAMES], count = 0;

int optimalVictim(int index);

int main() {
    printf("Enter the number of frames: ");
    scanf("%d", &numFrames);

    printf("Enter the number of reference strings: ");
    scanf("%d", &numReferences);

    printf("Enter the reference strings: ");
    for (int i = 0; i < numReferences; i++)
        scanf("%d", &references[i]);

    for (int i = 0; i < numFrames; i++) {
        frames[i] = -1;
        optimalCal[i] = 0;
    }

    printf("The Given string:\n");
    for (int i = 0; i < numReferences; i++)
        printf("%3d", references[i]);

    printf("\n\nOptimal Page Replacement Algorithm:\n\n");

    for (int i = 0; i < numReferences; i++) {
        int found = 0;
        printf("\nReference %d ->\t", references[i]);

        for (int j = 0; j < numFrames; j++) {
            if (frames[j] == references[i]) {
                found = 1;
                break;
            }
        }

        if (!found) {
            count++;
            if (count <= numFrames)
                victim++;
            else
                victim = optimalVictim(i);
            pageFaults++;
            frames[victim] = references[i];
            for (int j = 0; j < numFrames; j++)
                printf("%4d", frames[j]);
        }
    }

    printf("\nNumber of page faults: %d\n\n", pageFaults);

    return 0;
}

int optimalVictim(int index) {
    int temp, notFound;
    for (int i = 0; i < numFrames; i++) {
        notFound = 1;
        for (int j = index; j < numReferences; j++)
            if (frames[i] == references[j]) {
                notFound = 0;
                optimalCal[i] = j;
                break;
            }

        if (notFound == 1)
            return i;
    }

    temp = optimalCal[0];
    for (int i = 1; i < numFrames; i++)
        if (temp < optimalCal[i])
            temp = optimalCal[i];

    for (int i = 0; i < numFrames; i++)
        if (frames[temp] == frames[i])
            return i;
```

```
   return 0;
}
```

OUTPUT:



```
[it@itlab2sys136 13iii]$ gcc 13iii.c && ./a.out
................................
OPTIMAL PAGE REPLACEMENT ALGORITHN
.................................
Enter the no.of frames : 3
Enter the no.of reference string : 6
Enter the reference string : 6 5 4 3 2 1

................................
OPTIMAL PAGE REPLACEMENT ALGORITHM
................................

The Given string
....................
  6  5  4  3  2  1

ref no 6 ->        6   -1   -1
ref no 5 ->        6    5   -1
ref no 4 ->        6    5    4
ref no 3 ->        3    5    4
ref no 2 ->        2    5    4
ref no 1 ->        1    5    4
Number of page faults : 6
```

RESULT

Thus, the program for LFU was executed successfully.

FILE ORGANIZATION TECHNIQUES

Ex No: 14 (a) **SINGLE LEVEL DIRECTORY**

AIM: To write a c program to simulate Single level directory structure.

ALGORITHM

1. Start the program.
2. Get the number of main directories to be created.
3. Get the name of the directory and size of the directory.
4. Simulate the directory level and display the structure.
5. Stop the program.

PROGRAM

```
#include<stdio.h>
main()
{
int master,s[20];
char f[20][20][20];
char d[20][20];
int i,j;
printf("Enter number of directorios:");
scanf("%d",&master);
printf("Enter names of directories:");
for(i=0;i<master;i++)
scanf("%s",&d[i]);
printf("Enter size of directories:");
for(i=0;i<master;i++)
scanf("%d",&s[i]);
printf("Enter the file names :");
for(i=0;i<master;i++)
for(j=0;j<s[i];j++)
scanf("%s",&f[i][j]);
printf("\n");
printf(" directory\tsize\tfilenames\n");
printf("*************************************************\n");
for(i=0;i<master;i++)
{
printf("%s\t\t%2d\t",d[i],s[i]);
for(j=0;j<s[i];j++)
printf("%s\n\t\t\t",f[i][j]);
printf("\n");   }
printf("\t\n");
}
```

OUTPUT

```
[user@sys108 ~]$ vi singlelevel.c
[user@sys108 ~]$ cc singlelevel.c
[user@sys108 ~]$ ./a.out
 Enter number of directorios:2
 Enter names of directories: hai  hello
 Enter size of directories:2 2
 Enter the file names :sample   test    fibonacci      fact
  directory      size    filenames
 *************************************************
 hai              2          sample
                             test
 hello          2          fibonacci
                            fact
```

Ex.No: 14(b) TWO LEVEL DIRECTORY

AIM: To write a C program to simulate Two Level directory structures.

ALGORITHM

1. Start the program.
2. Get the number of main directories to be created, name of the directory and size of the directory.
3. Get the number of sub directories to be created, name of the directory and size of the directory.
4. Simulate the directory level and display the structure.
5. Stop the program.

PROGRAM

```
#include<stdio.h>
struct st
{
char dname[10];
char sdname[10][10];
char fname[10][10][10];
int ds,sds[10];
}dir[10];
void main()
{
int i,j,k,n;
printf("Enter number of directories:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter directory %d names:",i+1);
scanf("%s",&dir[i].dname);
printf("Enter size of directories:");
scanf("%d",&dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("Enter subdirectory name and size:");
scanf("%s",&dir[i].sdname[j]);
scanf("%d",&dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
{
printf("Enter file name:");
scanf("%s",&dir[i].fname[j][k]);
}
}
}
printf("\ndirname\t\tsize\tsubdirname\tsize\tfiles");
printf("\n*****************************************************\n");
for(i=0;i<n;i++)
{
printf("%s\t\t%d",dir[i].dname,dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("\t%s\t\t%d\t",dir[i].sdname[j],dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
printf("%s\t",dir[i].fname[j][k]);
printf("\n\t\t");
}
printf("\n");    }
}
```

OUTPUT

```
[user@sys108 ~]$ vi twolevel.c
[user@sys108 ~]$ cc twolevel.c
[user@sys108 ~]$ ./a.out
 Enter number of directories:2

 Enter directory 1 names:colleges

 Enter size of directories:1

 Enter subdirectory name and size:deemed 1

 Enter file name:stjosephs

 Enter directory 2 names:companies

 Enter size of directories:1

 Enter subdirectory name and size:MNC 1

 Enter file name:CTS

 dirname       size   subdirname        size    files

 *****************************************************

 colleges          1      affiliated            1      stjosephs


 companies         1        MNC                 1      CTS
```

Ex.No:14(c) **HIERARCHICAL & DAG**

AIM: To write a C program to implement Hierarchical and DAG.

ALGORITHM

1. Start the program.
2. Get the number of main directories to be created, name of the directory and size of the directory.
3. Get the number of sub directories to be created, name of the directory and size of the directory.
4. Get the name of the file to be shared between directories.
5. Simulate the directed acyclic graph and display the structure.
6. Stop the program.

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MIN_PER_RANK 1
#define MAX_PER_RANK 5
#define MIN_RANKS 3
#define MAX_RANKS 5
#define PERCENT 30
int main (void)
{
 int i, j, k,nodes = 0;
 srand (time (NULL));
 int ranks = MIN_RANKS
+ (rand () % (MAX_RANKS - MIN_RANKS + 1));
 printf ("digraph {\n");
for (i = 0; i < ranks; i++)
 {
    int new_nodes = MIN_PER_RANK
   +  (rand () % (MAX_PER_RANK - MIN_PER_RANK + 1));
  for (j = 0; j < nodes; j++)
  for (k = 0; k < new_nodes; k++)
  if ( (rand () % 100) < PERCENT)
  printf ("  %d -> %d;\n", j, k + nodes);
  nodes += new_nodes;
   }
 printf ("}\n");
 return 0;
}
```

OUTPUT

```
[user@sys108 ~]$ cc dgraph.c
[user@sys108 ~]$ ./a.out

digraph {

  0 -> 5;

  1 -> 6;

  2 -> 5;

  2 -> 8;

  3 -> 5;

  3 -> 8;

  4 -> 6;

  4 -> 7;

  1 -> 9;

  3 -> 9;

  6 -> 9;
```

Ex. No:15(a) **FCFS DISK SCHEDULING**

AIM: To write a c program to simulate FCFS disk scheduling.

ALGORITHM

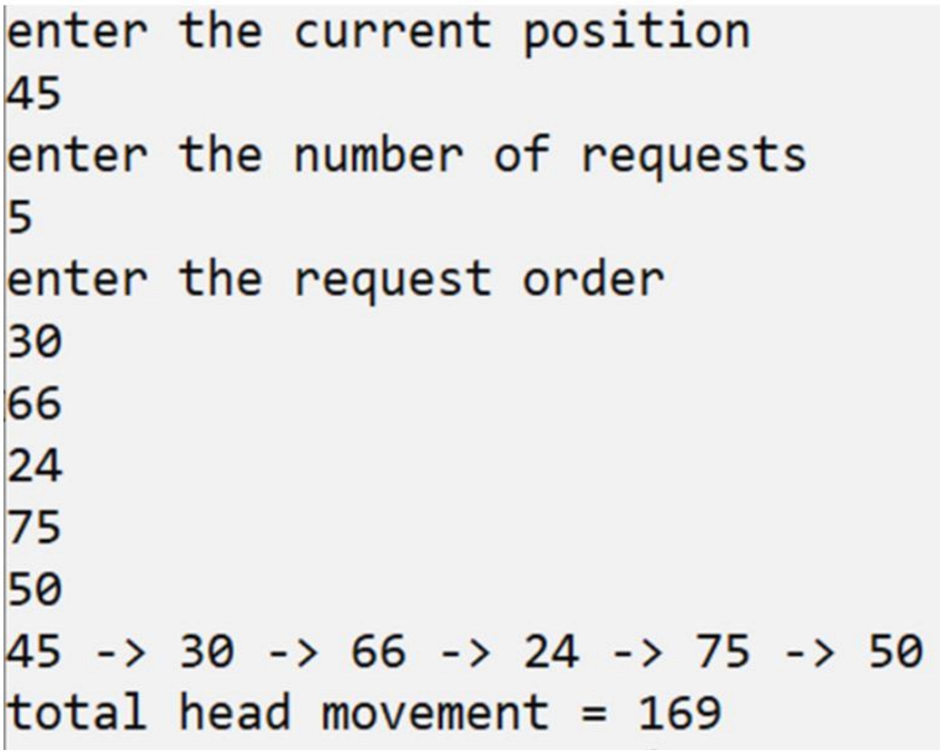
1. Enter current position.
2. Enter number of requests
3. Enter the request order.
4. Calculate the absolute value.
5. Calculate and display total head movement.

PROGRAM

```
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main()
{
int i,n,req[50],mov=0,cp;
printf("enter the current position\n");
scanf("%d",&cp);
printf("enter the number of requests\n");
scanf("%d",&n);
printf("enter the request order\n");
for(i=0;i<n;i++)
{
scanf("%d",&req[i]);
}
mov=mov+abs(cp-req[0]); // abs is used to calculate the absolute value
printf("%d -> %d",cp,req[0]);
for(i=1;i<n;i++)
{
mov=mov+abs(req[i]-req[i-1]);
printf(" -> %d",req[i]);
}
printf("\n");
```

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

OUTPUT



```
enter the current position
45
enter the number of requests
5
enter the request order
30
66
24
75
50
45 -> 30 -> 66 -> 24 -> 75 -> 50
total head movement = 169
```

Ex.No: 15(b) **SSTF DISK SCHEDULING**

AIM: To write a C program to simulate SSTF disk scheduling.

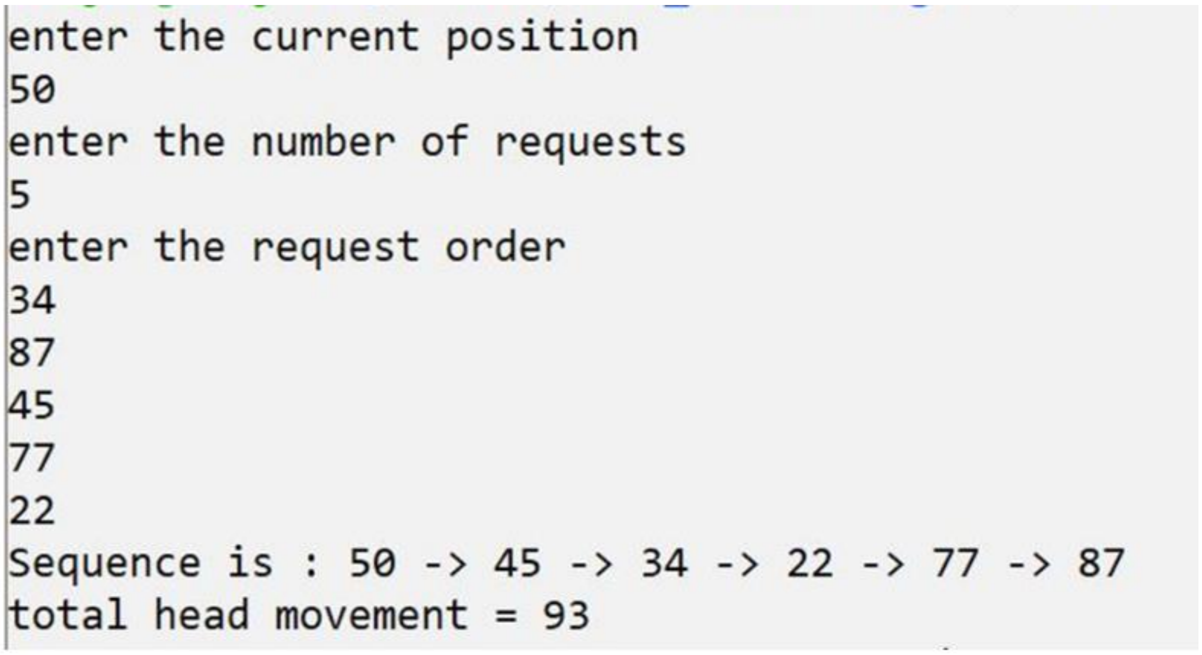
ALGORITHM

1. Get the index block number and number of files in the index block as input from user.
2. Get the file numbers (i.e referred block numbers holding file) as input.
3. Check whether that the input block number is already allocated if so, print block allocated.
4. Else increase the count and allocate the file.
5. Continue the loop to enter another index block.

PROGRAM

```
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main()
{
int i,n,k,req[50],mov=0,cp,index[50],min,a[50],j=0,mini,cp1;
printf("enter the current position\n");
scanf("%d",&cp);
printf("enter the number of requests\n");
scanf("%d",&n);
cp1=cp;
printf("enter the request order\n");
for(i=0;i<n;i++)
{
scanf("%d",&req[i]);
}
for(k=0;k<n;k++)
{
for(i=0;i<n;i++)
{
index[i]=abs(cp-req[i]); // calculate distance of each request from current position
}
// to find the nearest request
min=index[0];
mini=0;
for(i=1;i<n;i++)
{
if(min>index[i])
{
min=index[i];
mini=i;
}
}
a[j]=req[mini];
j++;
cp=req[mini]; // change the current position value to next request
req[mini]=999;
} // the request that is processed its value is changed so that it is not processed again
printf("Sequence is : ");
printf("%d",cp1);
mov=mov+abs(cp1-a[0]); // head movement
printf(" -> %d",a[0]);
for(i=1;i<n;i++)
{
mov=mov+abs(a[i]-a[i-1]); ///head movement
printf(" -> %d",a[i]);
} printf("\n");
printf("total head movement = %d\n",mov);
}
```

OUTPUT



```
enter the current position
50
enter the number of requests
5
enter the request order
34
87
45
77
22
Sequence is : 50 -> 45 -> 34 -> 22 -> 77 -> 87
total head movement = 93
```