Practical =1

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

int main() {
    int ch1;
    char msg[1000], ch;
    char old_name[20], new_name[20], source_file[20], target_file[20];
    FILE *fptr;

    printf("\n1. Create File and Write data\n2. Read the data\n3. Rename
File\n4. Copy Data of File to another\nEnter The Choice:");
    scanf("%d", &ch1);

    switch(ch1) {
        case 1:
            fptr = fopen("program2.txt", "w");
            if (fptr == NULL) {
                printf("Error!");
                exit(1);
            }
            printf("Enter message:");
            scanf(" %[^\n]", msg); // Modified to capture entire line
            fprintf(fptr, "%s", msg);
            fclose(fptr);
            break;
        case 2:
            fptr = fopen("program2.txt", "r");
            if (fptr == NULL) {
                printf("Error! opening file");
                exit(1);
            }
            fscanf(fptr, " %[^\n]", msg); // Modified to capture entire
line
            printf("Message Is: %s\n", msg);
            fclose(fptr);
            break;
        case 3:
            printf("\nEnter old program2.txt:");
            scanf("%s", old_name);
            printf("\nEnter new program2.txt:");
            scanf("%s", new_name);
            if (rename(old_name, new_name) == 0) {
                printf("File renamed successfully.\n");
            } else {
                printf("Unable to rename files. Please check files exist
and you have permission to modify files.\n");
            }
            break;
        case 4:
            printf("Enter name of file to copy: ");
            scanf("%s", source_file);
```

```
            FILE *source = fopen(source_file, "r");
            if (source == NULL) {
                printf("Error: Source file not found.\n");
                exit(EXIT_FAILURE);
            }
            printf("Enter name of target file: ");
            scanf("%s", target_file);
            FILE *target = fopen(target_file, "w");
            if (target == NULL) {
                printf("Error: Could not create target file.\n");
                fclose(source);
                exit(EXIT_FAILURE);
            }
            while ((ch = fgetc(source)) != EOF)
                fputc(ch, target);
            printf("File copied successfully.\n");
            fclose(source);
            fclose(target);
            break;
        default:
            printf("Invalid choice.\n");
            break;
    }

    return 0;
}

// Output :-
// 1. Create File and Write data
// 2.Read the data
// 3.Rename File
// 4.Copy Data of File to another
//  Enter The Choice:1
// Enter message: Hello
// 1. Create File and Write data
// 2.Read the data
// 3.Rename File
// 4.Copy Data of File to another
//  Enter The Choice:2
// Messege Is:=Hello
// 1. Create File and Write data
// 2.Read the data
// 3.Rename File
// 4.Copy Data of File to another
//  Enter The Choice:3
//  Enter old progam2.txt
// program2.txt
//  Enter new prog2.txt
// NewProgram.txt
// File renamed successfully.
// 1. Create File and Write data
// 2.Read the data
// 3.Rename File
// 4.Copy Data of File to another
```

```
//  Enter The Choice:4
// Enter name of file to copy
// program2.txt
// Enter name of target file
// NewProgram.txt
// File copied successfully.
```

```
#####################################################################

practical = 2

#include<stdio.h>
#define MAX 100

int main() {
    int Arrival_time[MAX], Burst_time[MAX], Completion_time[MAX],
    Turn_Around_time[MAX], Waiting_time[MAX],
    Average_Turn_Around_time = 0, Average_Waiting_time = 0, i, j;

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

    if (j <= 0 || j > MAX) {
        printf("Invalid number of processes.\n");
        return 1;
    }

    printf("Enter Arrival Time: ");
    for(i = 0; i < j; i++) {
        scanf("%d", &Arrival_time[i]);
    }

    printf("Enter Burst Time: ");
    for(i = 0; i < j; i++) {
        scanf("%d", &Burst_time[i]);
    }

    Completion_time[0] = Burst_time[0];
    for(i = 1; i < j; i++) {
        Completion_time[i] = Completion_time[i - 1] + Burst_time[i];
    }

    for(i = 0; i < j; i++) {
        Turn_Around_time[i] = Completion_time[i] - Arrival_time[i];
        Waiting_time[i] = Turn_Around_time[i] - Burst_time[i];
        Average_Waiting_time += Waiting_time[i];
        Average_Turn_Around_time += Turn_Around_time[i];
    }
```

```
    printf("\nProcess Arrival(T) Burst(T) Completion(T) Turn-Around(T)
Waiting(T)\n");
    for(i = 0; i < j; i++) {
        printf("P[%d]\t %d\t \t%d\t %d\t \t%d\t \t%d\n", i + 1,
               Arrival_time[i], Burst_time[i], Completion_time[i],
               Turn_Around_time[i], Waiting_time[i]);
    }

    printf("\nAverage Turn Around Time: %d\n", Average_Turn_Around_time /
j);
    printf("Average Waiting Time: %d\n", Average_Waiting_time / j);

    return 0;
}


// Output :-
// Enter Process U Want: 4
// Enter Arrival Time: 10 9
// 12
// 19 4
// 28
// Enter Burst Time: 20
// 12
// 34
// 5
// Process Arrival(T) Burst(T) Completion(T) Turn-Around(T) Waiting(T)
// P[1] 19 20 20 1 -19
// P[2] 12 12 32 20 8
// P[3] 14 34 66 52 18
// P[4] 28 5 71 43 38
// Average Turn Around Time: 29
// Average Waiting Time: 11
```

Pesudo code:

1- Input the processes along with their burst time(bt).
2- Find waiting time (wt) for all processes.
3- As first process that comes need not to wait so
4- waiting time for process 1 will be 0 i.e. wt[0] = 0.
5- Find waiting timefor all other processes i.e.for
a. process i->
b. wt[i] = bt[i-1] + wt[i-1].
6- Find turnaround time= waiting_time + burst_time
7- for all processes.
8- Find average waiting time=
1. total_waiting_time / no_of_processes.
9- Similarly, find average turnaround time=
1. total_turn_around_time / no_of_processes.

```
##################################################################


Practical =3

#include<stdio.h>
#include<conio.h>
void main()
{
int buffer[10], bufsize, in, out, produce, consume,
choice=0; in = 0;
out = 0;
bufsize = 10;
while(choice !=3)
{
printf("\n 1. Produce \t 2. Consume \t 3. Exit");
printf("\n Enter your choice: ");
scanf("%d", &choice);
switch(choice)
{
case 1: if((in+1)%bufsize==out)
printf("\nBuffer is Full");
else
{
printf("\nEnter the value: ");
scanf("%d", &produce);
buffer[in] = produce;
in = (in+1)%bufsize;
}
break;
case 2: if(in == out)
printf("\nBuffer is Empty");
else
{
consume = buffer[out];
printf("\nThe consumed value is %d", consume);
out = (out+1)%bufsize;
}
break;
}
}
}


// Output:-
// 1. Produce 2. Consume 3. Exit
//  Enter your choice: 1
// Enter the value: 10
//  1. Produce 2. Consume 3. Exit
//  Enter your choice: 2
// The consumed value is 10
//  1. Produce 2. Consume 3. Exit
//  Enter your choice: 1
// Enter the value: 30
```

```
//  1. Produce 2. Consume 3. Exit
//  Enter your choice: 2
// The consumed value is 30
//  1. Produce 2. Consume 3. Exit
//  Enter your choice: 3
```

The following is the pseudo-code for the producer:

```
void producer( ) {
while(T) {
produce ( )
wait (E)
wait (S)
append ( )
signal (S)
signal (F)
}
}
```

The following is the code for the consumer:

```
void consumer( ) {
while(T) {
wait(F)
wait(S)
take( )
signal(S)
signal(E)
use( )
}
}
```

##

Practical =4

```
#include <stdio.h>
#include <conio.h>
#define max 25

void main() {
    int frag[max], b[max], f[max], i, j, nb, nf, temp;
    static int bf[max], ff[max];

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

    printf("\nEnter the size of the blocks:-\n");
    for (i = 1; i <= nb; i++) {
        printf("Block %d:", i);
```

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

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

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

    printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment");
    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]);
    }
    getch();
}


// Output:
// Memory Management Scheme - First Fit Enter the number of blocks:5
// Enter the number of files:3
// Enter the size of the blocks:-
// Block 1:2
// Block 2:1
// Block 3:1
// Block 4:1
// Block 5:2
// Enter the size of the files :-
// File 1:2
// File 2:1
// File 3:3
// File_no: File_size : Block_no: Block_size: Fragement
// 1 2 1 2 0
// 2 1 2 1 0
// 3 3 0 1 -1


############################################################
Parctical =5
```

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

struct fileTable {
    char name[20];
    int sb, nob;
} ft[30];

int main() {
    int i, j, n;
    char s[20];

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

    for (i = 0; i < n; i++) {
        printf("\nEnter file name %d: ", i + 1);
        scanf("%s", ft[i].name);
        printf("Enter starting block of file %d: ", i + 1);
        scanf("%d", &ft[i].sb);
        printf("Enter number of blocks in file %d: ", i + 1);
        scanf("%d", &ft[i].nob);
    }

    printf("\nEnter the file name to be searched: ");
    scanf("%s", s);

    for (i = 0; i < n; i++) {
        if (strcmp(s, ft[i].name) == 0)
            break;
    }

    if (i == n)
        printf("\nFile Not Found");
    else {
        printf("\nFILE NAME\tSTART BLOCK\tNO OF BLOCKS\tBLOCKS
OCCUPIED\n");
        printf("%s\t\t%d\t\t%d\t\t", ft[i].name, ft[i].sb, ft[i].nob);
        for (j = 0; j < ft[i].nob; j++)
            printf("%d, ", ft[i].sb + j);
    }

    getch();
    return 0;
}

// INPUT: Enter no of files :3
// Enter file name 1 :A
// Enter starting block of file 1 :85 Enter no of blocks in file 1 :6
// Enter file name 2 :B
// Enter starting block of file 2 :102
// Enter no of blocks in file 2 :4
// Enter file name 3 :C
```

```
// Enter starting block of file 3 :60 Enter no of blocks in file 3 :4
// Enter the file name to be searched -- B
// OUTPUT:
// FILE NAME START BLOCK NO OF BLOCKS BLOCKS OCCUPIED
// B 102 4 102, 103, 104, 105
```

Algorithm for Sequential File Allocation:

Step 1: Start the program.

Step 2: Get the number of memory partition and their sizes.

Step 3: Get the number of processes and values of block size for each process.

Step 4: First fit algorithm searches all the entire memory block until a hole which is big enough is encountered. It allocates that memory block for the requesting process.

Step 5: Best-fit algorithm searches the memory blocks for the smallest hole which can be allocated to requesting process and allocates it.

Step 6: Worst fit algorithm searches the memory blocks for the largest hole and allocates it to the process.

Step 7: Analyses all the three memory management techniques and display the best algorithm which utilizes the memory resources effectively and efficiently.

Step 8: Stop the program.

##

practical =6

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

struct directory {
    char dname[10];
    char fname[10][10];
    int fcnt;
} dir;

void main() {
    int i, ch;
    char f[30];
    dir.fcnt = 0;

    printf("\nEnter name of directory -- ");
    scanf("%s", dir.dname);

    while (1) {
        printf("\n\n1. Create File\t2. Delete File\t3. Search File\n4.
Display Files\t5. Exit\nEnter your choice -- ");
        scanf("%d", &ch);
```

```
        switch (ch) {
            case 1:
                printf("\nEnter the name of the file -- ");
                scanf("%s", dir.fname[dir.fcnt]);
                dir.fcnt++;
                break;
            case 2:
                printf("\nEnter the name of the file -- ");
                scanf("%s", f);
                for (i = 0; i < dir.fcnt; i++) {
                    if (strcmp(f, dir.fname[i]) == 0) {
                        printf("File %s is deleted ", f);
                        strcpy(dir.fname[i], dir.fname[dir.fcnt - 1]);
                        break;
                    }
                }
                if (i == dir.fcnt)
                    printf("File %s not found", f);
                else
                    dir.fcnt--;
                break;
            case 3:
                printf("\nEnter the name of the file -- ");
                scanf("%s", f);
                for (i = 0; i < dir.fcnt; i++) {
                    if (strcmp(f, dir.fname[i]) == 0) {
                        printf("File %s is found ", f);
                        break;
                    }
                }
                if (i == dir.fcnt)
                    printf("File %s not found", f);
                break;
            case 4:
                if (dir.fcnt == 0)
                    printf("\nDirectory Empty");
                else {
                    printf("\nThe Files are -- ");
                    for (i = 0; i < dir.fcnt; i++)
                        printf("\t%s", dir.fname[i]);
                }
                break;
            default:
                exit(0);
        }
    }
    getch();
}


// Output:
// Enter name of directory -- CSE
// 1. Create File 2. Delete File 3. Search File
```

```
// 4. Display Files 5. Exit Enter your choice â€“ 1 Enter the name of the
file -- A
// 1. Create File 2. Delete File 3. Search File
// 4. Display Files 5. Exit
// Enter your choice â€“ 1
// Enter the name of the file -- B
// 1. Create File 2. Delete File 3. Search File
// 4. Display Files 5. Exit Enter your choice â€“ 1
// Enter the name of the file -- C
// 1. Create File 2. Delete File 3. Search File
// 4. Display Files 5. Exit
// Enter your choice â€“ 4
// The Files are -- A B C 1. Create File 2. Delete File 3. Search File
// 4. Display Files 5. Exit
// Enter your choice â€“ 3
// Enter the name of the file â€“ ABC File ABC not found
// 1. Create File 2. Delete File 3. Search File
// 4. Display Files 5. Exit
// Enter your choice â€“ 2
// Enter the name of the file â€“ B
// File B is deleted
// 1. Create File 2. Delete File 3. Search File
// 4. Display Files 5. Exit
// Enter your choice â€“ 5 ...Program finished with exit code 0
```

Algorithm for Single Level Directory Structure:

```
Step 1: Start
Step 2: Initialize values gd=DETECT,gm,count,i,j,mid,cir_x;
Initialize character array fname[10][20];
Step 3: Initialize graph function as
Initgraph(& gd, &gm," c:/tc/bgi");
Clear device();
Step 4:set back ground color with setbkcolor();
Step 5:read number of files in variable count.
Step 6:if check i<count
Step 7: for i=0 & i<count
i increment;
Cleardevice();
setbkcolor(GREEN);
read filename;
setfillstyle(1,MAGENTA);
Step 8: mid=640/count;
cir_x=mid/3;
bar3d(270,100,370,150,0,0);
settextstyle(2,0,4);
settextstyle(1,1);outtextxy(320,125,
"rootdirectory");setcolor(BLUE);
i++;
Step 9:for j=0&&j<=i&&cir_x+=mid
j increment;
```

```
line(320,150,cir_x,250);
fillellipse(cir_x,250,30,30);
outtextxy(cir_x,250,fname[i]); Step
10: End


############################################################

practical =7

#include<stdio.h>
#include<conio.h>
main()
{
int i, j, k, f, pf=0, count=0, rs[25], m[10], n;
printf("\n Enter the length of reference string -- ");
scanf("%d",&n);
printf("\n Enter the reference string -- ");
for(i=0;i<n;i++)
scanf("%d",&rs[i]);
printf("\n Enter no. of frames -- ");
scanf("%d",&f);
for(i=0;i<f;i++)
m[i]=-1;
printf("\n The Page Replacement Process is -- \n");
for(i=0;i<n;i++)
{
for(k=0;k<f;k++)
{
if(m[k]==rs[i])
break;
}
if(k==f)
{
m[count++]=rs[i];
pf++;
}
for(j=0;j<f;j++)
printf("\t%d",m[j]); if(k==f)
printf("\tPF No. %d",pf); printf("\n"); if(count==f)
count=0;
}
printf("\n The number of Page Faults using FIFO are %d",pf);
getch();
}


// Output:
// Enter the length of reference string -- 5 Enter the reference string -
- 1 2 3 4 5 Enter no. of
// frames -- 4 The Page Replacement Process is --
// 1 -1 -1 -1 PF No. 1 1 2 -1 -1 PF No. 2 1 2 3 -1 PF No. 3 1 2 3 4 PF
No. 4 5 2 3 4 PF No. 5 The
// number of Page Faults using FIFO are 5
```

Procedure:
1- Start traversing the pages.
i) If set holds less pages thancapacity.
a) Insert page into the set one by oneuntil
the size of set reaches capacity or all
page requests areprocessed.
b) Simultaneously maintain the pages inthe
queue to performFIFO.
c) Increment pagefault
ii) Else
If current page is present in set, do nothing.
Else
a) Remove the first page from the queue
as it was the first to be enteredin
the memory
b) Replace the first page in the queuewith
the current page in thestring.
c) Store current page in the queue.
d) Increment pagefaults.
2. Return pagefaults.

##

Practical = 8

NAME:Rajashri V. Bari
CLASS: TE(A)
ROLL NO: 14
PRACTICAL NO:8
AIM: Write a C program to simulate FCFS disk scheduling
algorithm

```
#include<stdio.h>
main()
{
int t[20], n, I, j, tohm[20], tot=0;
float avhm;
clrscr();
printf("enter the no.of tracks");
scanf("%d",&n); printf("enter the tracks to be traversed");
for(i=2;i<n+2;i++)
scanf("%d",&t*i+); for(i=1;i<n+1;i++)
{
tohm[i]=t[i+1]-t[i]; if(tohm[i]<0) tohm[i]=tohm[i]*(-1); }
for(i=1;i<n+1;i++) tot+=tohm[i];
avhm=(float)tot/n;
printf("Tracks traversed\tDifference between tracks\n");
for(i=1;i<n+1;i++)
printf("%d\t\t\t%d\n",t*i+,tohm*i+); printf("\nAverage header
movements:%f",avhm); getch();
```

}
Output :-
INPUT
Enter no.of tracks:9
Enter track position:55 58 60 70 18 90 150 160 184 OUTPUT
Tracks traversed Difference between tracks
55
58
60
70
18
90
150
160
184
45
3
2
10
52
72
60
10
24
Average header movements: 30.888889

Algorithm:
1. Let Request array represents an array storing indexes of tracks that have been requested in
ascending order of their time of arrival. 'head' is the position of diskhead.
2. Let us one by one take the tracks in default order and calculate the absolute distance of the
track from thehead.
3. Increment the total seek count with thisdistance.
4. Currently serviced track position now becomes the new headposition.
5. Go to step 2 until all tracks in request array have not beenserviced.
Example:
Input:
Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}
Initial head position = 50
Output:
Total number of seek operations = 510
Seek Sequence is
176, 79, 34, 60, 92, 11, 41, 114

Therefore, the total seeks count is calculated as:
= (176-50)+(176-79)+(79-34)+(60-34)+(92-60)+(92-11)+(41-11)+(114-41)
= 510
//