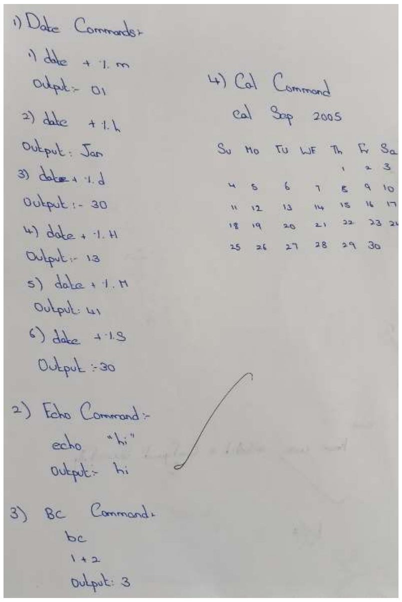
Ex No: 1
Date: 23.01.2025

BASIC LINUX COMMANDS



1) Date Commands:-

```
1) date + %m
 Output:- 01
2) date + %h
Output: Jan
3) date + %d
Output:- 30
4) date + %H
Output:- 13
5) date + %M
 Output: 41
6) date + %S
 Output :-30
```

2) Echo Command:-

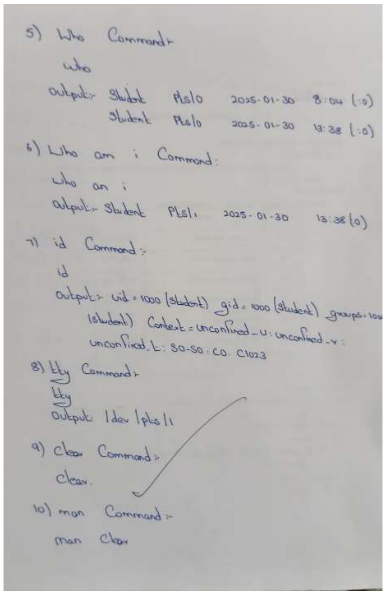
```
echo "hi"
Output:- hi
```

3) Bc Command:-

```
bc
1 + 2
Output: 3
```

4) Cal Command

```
cal Sep 2005

Su  Mo  Tu  WE  Th  Fr  Sa
                 1   2   3
4   5   6   7   8   9   10
11  12  13  14  15  16  17
18  19  20  21  22  23  24
25  26  27  28  29  30
```



5) Who Command:-

```
who
Output:- Student  Pts/0   2025-01-30  8:04 (:0)
         Student  Pts/0   2025-01-30  13:38 (:0)
```

6) Who am i Command:

```
who am i
Output:- Student  Pts/1   2025-01-30   13:38 (0)
```

7) id Command:-

```
id
Output:- uid=1000(student) gid=1000(student) groups=100
    (student) Context=unconfined_u:unconfined_r:
    unconfined_t:S0-S0:C0.C1023
```

8) tty Command:-

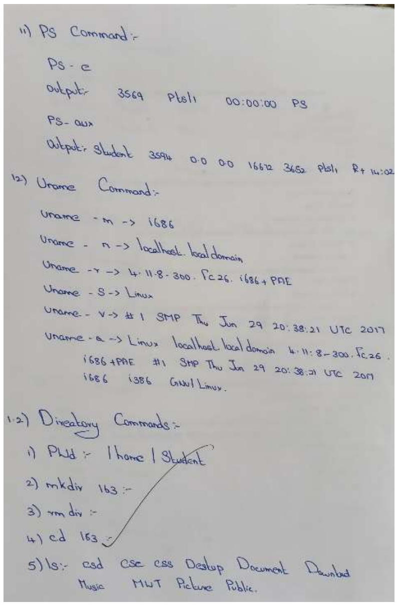
```
tty
Output: /dev/pts/1
```

9) clear Command:-

```
clear.
```

10) man Command:-

```
man Clear
```



11) PS Command:-

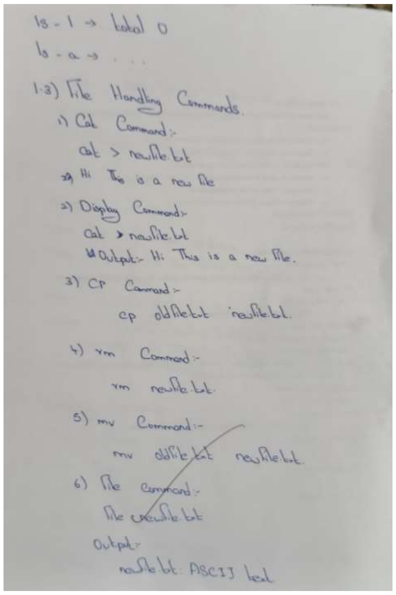
```
PS -e
Output:-   3569   Pts/1   00:00:00  PS
PS -aux
Output:- Student  3594  0.0  0.0  16672  3652  Pts/1  R+  14:02
```

12) Uname Command:-

```
Uname -m -> i686
Uname -n -> localhost.localdomain
Uname -r -> 4.11.8-300.fc26.i686+PAE
Uname -S -> Linux
Uname -v -> #1 SMP Thu Jun 29 20:38:21 UTC 2017
Uname -a -> Linux localhost.localdomain 4.11.8-300.fc26.
      i686+PAE  #1 SMP Thu Jun 29 20:38:21 UTC 2017
      i686  i386  GNU/Linux.
```

1.2) Directory Commands:-

```
1) PWd :- /home/Student
2) mkdir 163 :-
3) rm dir :-
4) cd 163 :-
5) ls:- csd  cse  css  Destop  Document  Download
        Music  MWT  Picture  Public.
```



```
ls -l -> total 0
ls -a -> . . .
```

1.3) File Handling Commands.

1) Cat Command:-

```
cat > newfile.txt
Hi This is a new file
```

2) Display Command:-

```
cat > newfile.txt
Output:- Hi This is a new file.
```

3) CP Command:-

```
cp oldfile.txt newfile.txt.
```

4) rm Command:-

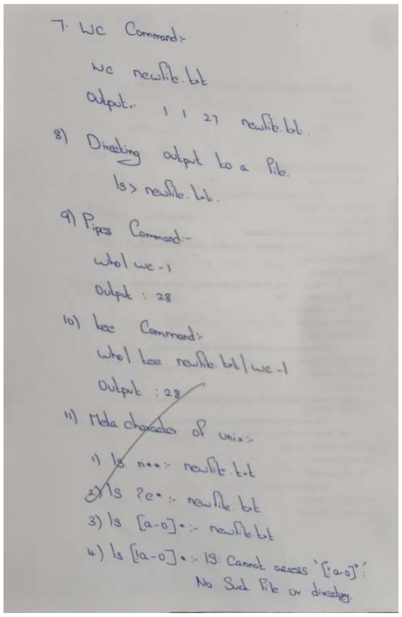
```
rm newfile.txt.
```

5) mv Command:-

```
mv oldfile.txt newfile.txt.
```

6) file Command:-

```
file newfile.txt
Output:-
  newfile.txt: ASCII text
```



7. Wc Command:-

```
wc newfile.txt
Output:-  1 1 27 newfile.txt.
```

8) Directing output to a file.

```
ls> newfile.txt.
```

9) Pipes Command:-

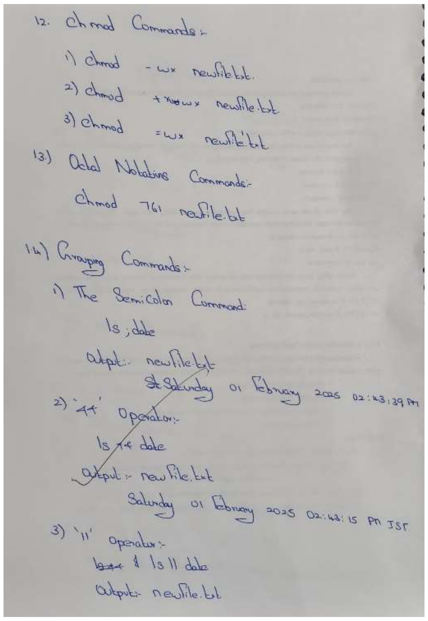
```
who| wc -1
Output : 28
```

10) tee Command:-

```
who| tee newfile.txt| wc -1
Output : 28
```

11) Meta characters of unix:-

```
1) ls n**:- newfile.txt
2) ls ?e*:- newfile.txt
3) ls [a-o]*:- newfile.txt
4) ls [!a-o]*:- ls: Cannot access '[!a-o]*':
                No Such file or directory.
```



12. Chmod Commands:-

```
1) chmod  -wx newfile.txt.
2) chmod  +~~rw~~wx newfile.txt.
3) chmod  =wx newfile.txt.
```

13.) Octal Notations Commands:-

```
chmod 761 newfile.txt
```

14) Grouping Commands:-

1) The Semicolon Command:

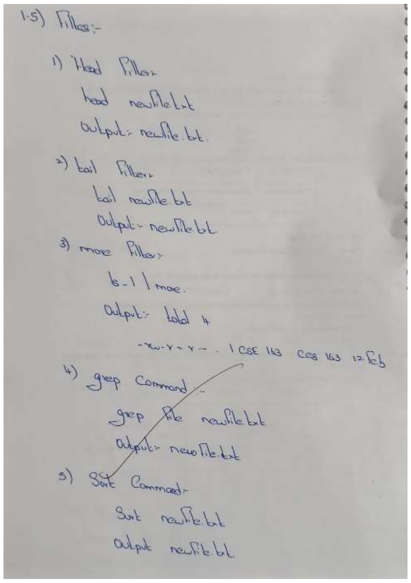
```
ls ; date
Output:- newfile.txt
         ~~St~~ Saturday 01 February 2025 02:43:39 PM
```

2) '&&' Operator:-

```
ls && date
Output:- newfile.txt
         Saturday 01 February 2025 02:43:15 PM IST
```

3) '||' Operator:-

```
~~ls&&~~ & ls || date
Output:- newfile.txt
```



1.5) Filters:-

1) Head Filter:-

```
head newfile.txt
Output:- newfile.txt.
```

2) tail Filter:-

```
tail newfile.txt
Output:- newfile.txt.
```

3) more Filter:-

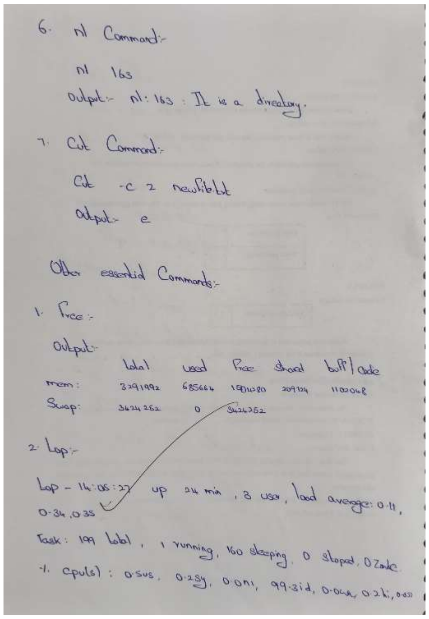
```
ls -1 | more.
Output:- total 4
         -rw-r--r-- . 1 CSE 163  CES 163  12 Feb
```

4) grep Command:-

```
grep file newfile.txt
Output:- newfile.txt
```

5) Sort Command:-

```
Sort newfile.txt
Output newfile.txt.
```



6. nl Command:-

```
nl 163
Output:- nl: 163 : It is a directory.
```

7. Cut Command:-

```
Cut -c 2 newfile.txt
Output:- e
```

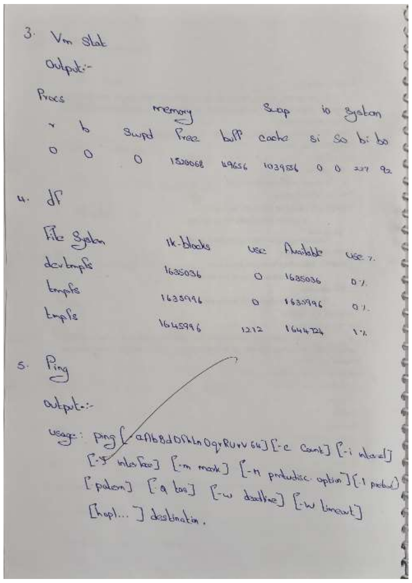
Other essential Commands:-

1. free:-

```
Output:-
          total     used     free      shared   buff/cache
mem:      3291992   685664   1504280   209124   1102048
Swap:     3424252   0        3424252
```

2. top:-

```
top - 14:05:27 up 24 min , 3 user, load average: 0.11,
0.34, 0.35
Task: 199 total , 1 running, 160 sleeping. 0 stopped, 0 zombie.
%. cpu(s) : 0.5us, 0.2sy, 0.0ni, 99.3id, 0.0wa, 0.2hi, 0.0si
```



3. Vm Stat

Output:-

```
Procs
                memory              Swap      io  system
 r    b    Swpd   Free      buff   cache    si  So  bi  bo
 0    0     0    1520068   49656  1039536   0   0   227 92
```

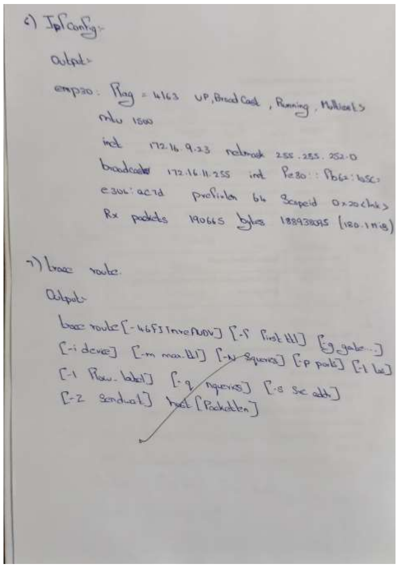
4. df

```
File System      1k-blocks     use    Available    use %
devtmpfs         1635036        0     1635036       0%
tmpfs            1635996        0     1635996       0%
tmpfs            1645996      1272    1644724       1%
```

5. Ping

output:-

```
usage: ping [-aAbBdDfhLnOqrRUvV64] [-c Count] [-i interval]
       [-I interface] [-m mark] [-M pmtudisc option] [-l preload]
       [pattern] [-Q tos] [-w deadline] [-W timeout]
       [hop1...] destination.
```



6) Ipconfig:-

Output:-

```
emp30: flag = 4163 UP,BroadCast, Running, Multicast>
       mtu 1500
       inet    172.16.9.23  netmask 255.255.252.0
       broadcast  172.16.11.255   inet  fe80::fb62:b5c:
       e304:ac7d    prefixlen 64  Scopeid 0x20<link>
       Rx  packets  190665  bytes  188938095 (180.1 mib)
```

7) trace route.

Output:-

```
trace route [-46FJInreAUDV] [-f first ttl] [-g gate...]
[-i device] [-m max. ttl] [-N Squeries] [-p port] [-t tos]
[-l flow_label] [-q nqueries] [-s Src addr]
[-z sendwait] host [Packetlen]
```

Result:

The Unix/Linux commands successfully displayed system information, managed files, and performed navigation tasks, confirming the proper setup and functionality of the environment.

Ex. no: 2a)
Date: 30.01.2025

SHELL SCRIPT

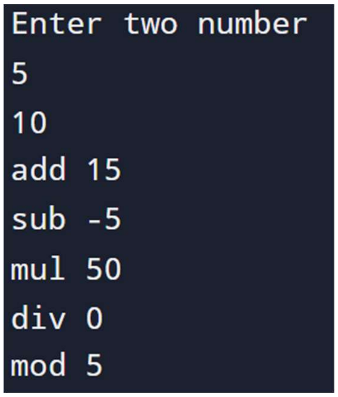
Aim:

To write a Shell script to display basic calculator.

Program:

```
echo "Enter two number"
read a
read b

echo "add $((a + b))"
echo "sub $((a - b))"
echo "mul $((a * b))"
echo "div $((a / b))"
echo "mod $((a % b))"0;
}
```

Output:



```
Enter two number
5
10
add 15
sub -5
mul 50
div 0
mod 5
```

Result:

Hence, the basic calculator program was executed successfully.

Ex. no: 2b)
Date: 30.01.2025

SHELL SCRIPT

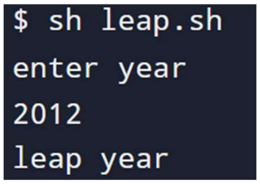
Aim:

To write a Shell script to test given year is leap or not using conditional statement.

Program:

```
echo enter year
read y
if [ $((y%4)) -eq 0 -a $((y%100)) -ne 0 -o $((y%400)) -eq 0 ]
then echo leap year
else echo not leap year
fi
```

Output:



```
$ sh leap.sh
enter year
2012
leap year
```

Result:

Hence, the Shell Script to check leap year was executed successfully.

Ex. no: 3a)

Date: 01.02.2025

SHELL SCRIPT – REVERSE OF DIGIT

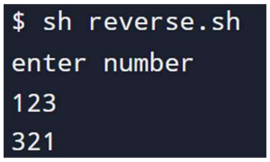
Aim:

To write a Shell script to reverse a given digit using looping statement.

Program:

```
echo enter number
read n
rev=0
while [ $n -gt 0 ]
do
  digit=$((n%10))
  rev=$((rev*10+digit))
  n=$((n/10))
done
echo $rev
```

Output:



```
$ sh reverse.sh
enter number
123
321
```

Result:

Hence, the Shell Script to reverse the given digit was executed successfully.

Ex. no: 3b)
Date: 01.02.2025

SHELL SCRIPT – FIBBONACCI SERIES

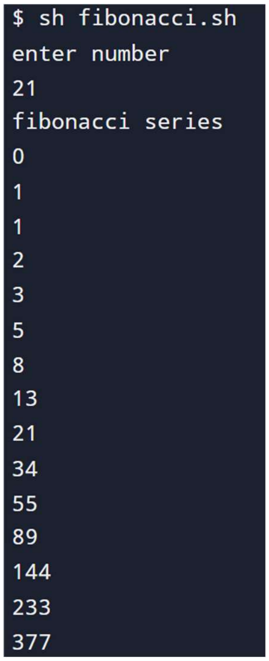
Aim:

To write a Shell script to generate a Fibonacci series using for loop.

Program:

```
echo enter number
read n
a=0
b=1
echo "fibonacci series"
for (( i=0; i<n; i++ ))
do
  echo $a
  fn=$((a+b))
  a=$b
  b=$fn
done
```

Output:



```
$ sh fibonacci.sh
enter number
21
fibonacci series
0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
```

Result:

Hence, the Shell Script to generate the Fibonacci series was executed successfully.

Ex. No.: 4a)
Date: 13.02.2025

EMPLOYEE AVERAGE PAY

Aim:

To find out the average pay of all employees whose salary is more than 6000 and no. of days worked is more than 4.

Program:

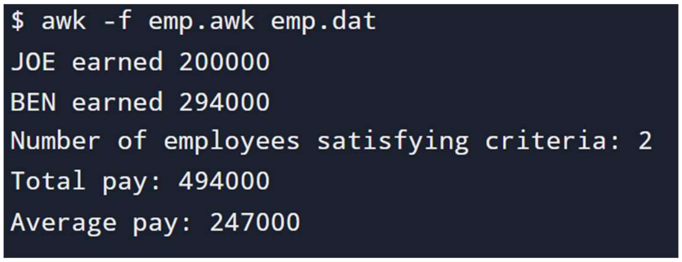
Create the emp.dat file:

```
JOE 40000 5
BEN 49000 6
AMY 39000 4
```

Create the emp.awk script:

```
BEGIN {
  totalPay = 0
  count = 0
}
{
  if ($2 > 6000 && $3 > 4) {
    print $1, "earned", $2 * $3
    totalPay += $2 * $3
    count++
  }
}
END {
  if (count > 0) {
    print "Number of employees satisfying criteria:", count
    print "Total pay:", totalPay
    print "Average pay:", totalPay / count
  } else {
    print "No employees satisfy the criteria."
  }
}
```

Output:



```
$ awk -f emp.awk emp.dat
JOE earned 200000
BEN earned 294000
Number of employees satisfying criteria: 2
Total pay: 494000
Average pay: 247000
```

Result:

Hence, the Shell script to calculate the average pay of employees was executed successfully, and the average pay was calculated correctly.

Ex. No.: 4b)
Date: 13.02.2025

RESULTS OF EXAMINATION

Aim:

To print the pass/fail status of a student in a class

Program:

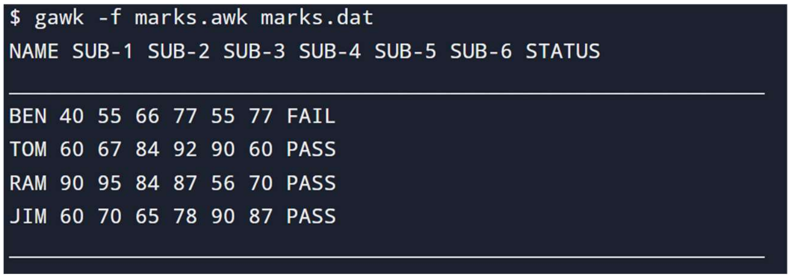
Create the marks.dat file (student marks data):

```
BEN 40 55 66 77 55 77
TOM 60 67 84 92 90 60
RAM 90 95 84 87 56 70
JIM 60 70 65 78 90 87
```

Create the marks.awk script:

```
BEGIN {
  print "NAME SUB-1 SUB-2 SUB-3 SUB-4 SUB-5 SUB-6 STATUS"
  print "____________________________________________________________"
}
{
  status = "PASS"
  for (i = 2; i <= 7; i++) {
    if ($i < 45) {
      status = "FAIL"
      break
    }
  }
  print $1, $2, $3, $4, $5, $6, $7, status
}
```

Output:



```
$ gawk -f marks.awk marks.dat
NAME SUB-1 SUB-2 SUB-3 SUB-4 SUB-5 SUB-6 STATUS
____________________________________________________________
BEN 40 55 66 77 55 77 FAIL
TOM 60 67 84 92 90 60 PASS
RAM 90 95 84 87 56 70 PASS
JIM 60 70 65 78 90 87 PASS
____________________________________________________________
```

Result:

The Shell script to determine the pass/fail status based on the subject marks was executed successfully.

Ex. No.: 5
Date: 19.02.2025

SYSTEM CALLS PROGRAMMING

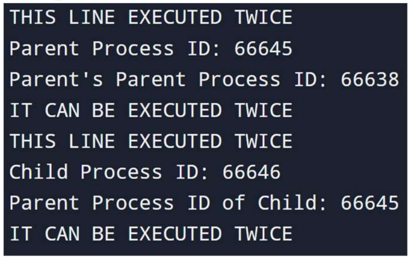
Aim:

To experiment system calls using fork(), execlp() and pid() functions.

Program:

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

int main() {
    int pid;
    pid = fork();
    if (pid == -1) {
        printf("CHILD PROCESS NOT CREATED\n");
        exit(0);
    }
    printf("THIS LINE EXECUTED TWICE\n");
    if (pid == 0) {
        printf("Child Process ID: %d\n", getpid());
        printf("Parent Process ID of Child: %d\n", getppid());
    }
    else {
        printf("Parent Process ID: %d\n", getpid());
        printf("Parent's Parent Process ID: %d\n", getppid());
    }
    printf("IT CAN BE EXECUTED TWICE\n");
    return 0;
}
```

Output:



```
THIS LINE EXECUTED TWICE
Parent Process ID: 66645
Parent's Parent Process ID: 66638
IT CAN BE EXECUTED TWICE
THIS LINE EXECUTED TWICE
Child Process ID: 66646
Parent Process ID of Child: 66645
IT CAN BE EXECUTED TWICE
```

Result:

The system calls fork(), getpid(), and getppid() were successfully used to create a child process, print process details, and show that both parent and child execute the same code.

Ex. No.: 6a)
Date: 20.02.2025

FIRST COME FIRST SERVE

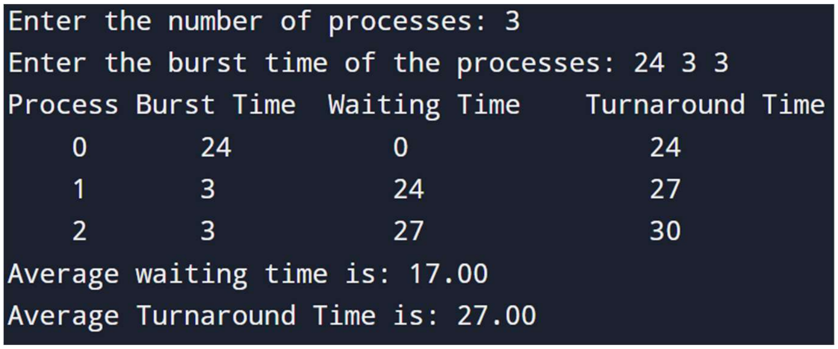
Aim:

To implement First-come First- serve (FCFS) scheduling.

Program:

```
#include <stdio.h>
int main() {
    int n,i,j,bt[10],wt[10],tat[10],total_wt=0,total_tat=0;
    printf("Enter the number of processes: ");
    scanf("%d",&n);
    printf("Enter the burst time of the processes: ");
    for(i=0;i<n;i++) scanf("%d",&bt[i]);
    wt[0]=0;
    for(i=1;i<n;i++) wt[i]=bt[i-1]+wt[i-1];
    for(i=0;i<n;i++) tat[i]=bt[i]+wt[i];
    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for(i=0;i<n;i++) printf("%d\t%d\t\t%d\t\t%d\n",i,bt[i],wt[i],tat[i]);
    for(i=0;i<n;i++) {
        total_wt+=wt[i];
        total_tat+=tat[i];
    }
    printf("Average waiting time is: %.2f\n",(float)total_wt/n);
    printf("Average Turnaround Time is: %.2f\n",(float)total_tat/n);
    return 0;
}
```

Output:



```
Enter the number of processes: 3
Enter the burst time of the processes: 24 3 3
Process Burst Time  Waiting Time    Turnaround Time
    0       24          0               24
    1       3           24              27
    2       3           27              30
Average waiting time is: 17.00
Average Turnaround Time is: 27.00
```

Result:

The program implements the First-Come-First-Serve (FCFS) scheduling technique, calculating the waiting time, turnaround time, and averages and executed successfully.

Ex. No.: 6b)
Date: 26.02.2025

SHORTEST JOB FIRST

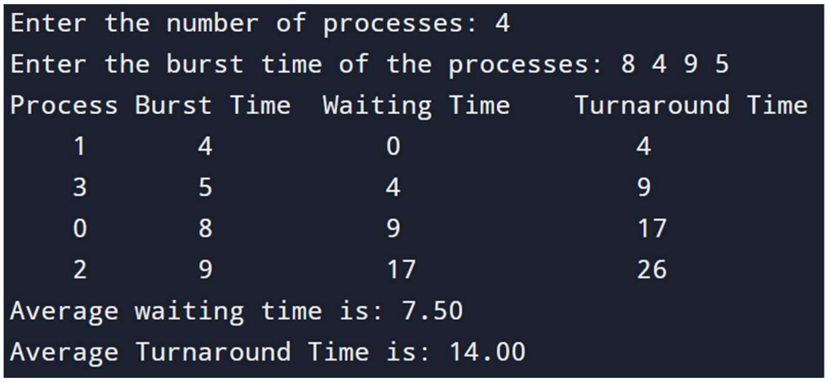
Aim:

To implement the Shortest Job First (SJF) scheduling.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int n,i,j;
    printf("Enter the number of processes: ");
    scanf("%d",&n);
    int burst_time[n],waiting_time[n],turnaround_time[n],pid[n];
    int total_wt=0,total_tat=0;
    printf("Enter the burst time of the processes: ");
    for(i=0;i<n;i++){
        pid[i]=i;
        scanf("%d",&burst_time[i]);
        waiting_time[i]=0;
        turnaround_time[i]=0;
    }
    for(i=0;i<n-1;i++){
        for(j=i+1;j<n;j++){
            if(burst_time[i]>burst_time[j]){
                int temp=burst_time[i];
                burst_time[i]=burst_time[j];
                burst_time[j]=temp;
                temp=pid[i];
                pid[i]=pid[j];
                pid[j]=temp;
            }
        }
    }
    for(i=1;i<n;i++){
        waiting_time[i]=burst_time[i-1]+waiting_time[i-1];
    }
    for(i=0;i<n;i++){
        turnaround_time[i]=burst_time[i]+waiting_time[i];
    }
    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for(i=0;i<n;i++){
        printf("%d\t%d\t\t%d\t\t%d\n",pid[i],burst_time[i],waiting_time[i],turnaround_time[i]);
    }
    for(i=0;i<n;i++){
        total_wt+=waiting_time[i];
        total_tat+=turnaround_time[i];
    }
    printf("Average waiting time is: %.2f\n",(float)total_wt/n);
    printf("Average Turnaround Time is: %.2f\n",(float)total_tat/n);
    return 0;
}
```

Output:



```
Enter the number of processes: 4
Enter the burst time of the processes: 8 4 9 5
Process Burst Time  Waiting Time    Turnaround Time
    1       4           0               4
    3       5           4               9
    0       8           9               17
    2       9           17              26
Average waiting time is: 7.50
Average Turnaround Time is: 14.00
```

Result:

The program implements the Shortest Job First (SJF) scheduling technique, calculating the waiting time, turnaround time, and averages, and executed successfully.

Ex. No.: 6c)
Date: 27.02.2025

PRIORITY SCHEDULING

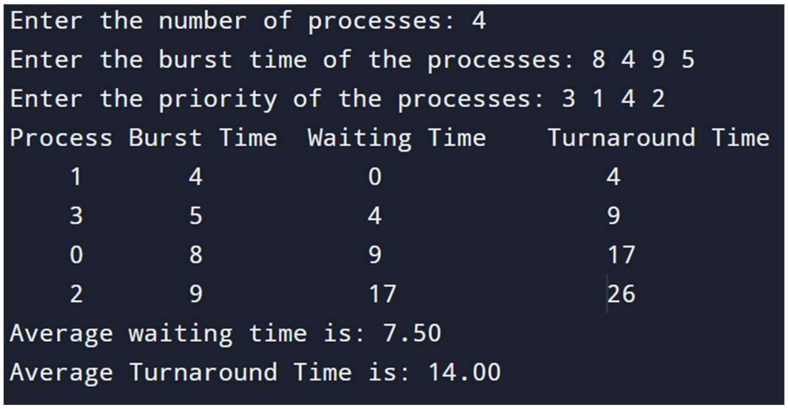
Aim:

To implement priority scheduling technique.

Program:

```
#include <stdio.h>
int main(){
    int n,i,j;
    printf("Enter the number of processes: ");
    scanf("%d",&n);
    int bt[n],wt[n],tat[n],p[n],pri[n],total_wt=0,total_tat=0;
    printf("Enter the burst time of the processes: ");
    for(i=0;i<n;i++){
        scanf("%d",&bt[i]);
        p[i]=i;
    }
    printf("Enter the priority of the processes: ");
    for(i=0;i<n;i++) scanf("%d",&pri[i]);
    for(i=0;i<n-1;i++){
        for(j=i+1;j<n;j++){
            if(pri[i]>pri[j]){
                int temp=pri[i];pri[i]=pri[j];pri[j]=temp;
                temp=bt[i];bt[i]=bt[j];bt[j]=temp;
                temp=p[i];p[i]=p[j];p[j]=temp;
            }
        }
    }
    wt[0]=0;
    for(i=1;i<n;i++) wt[i]=bt[i-1]+wt[i-1];
    for(i=0;i<n;i++) tat[i]=bt[i]+wt[i];
    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for(i=0;i<n;i++){
        printf("%d\t%d\t\t%d\t\t%d\n",p[i],bt[i],wt[i],tat[i]);
        total_wt+=wt[i];
        total_tat+=tat[i];
    }
    printf("Average waiting time is: %.2f\n",(float)total_wt/n);
    printf("Average Turnaround Time is: %.2f\n",(float)total_tat/n);
    return 0;
}
```

Output:



```
Enter the number of processes: 4
Enter the burst time of the processes: 8 4 9 5
Enter the priority of the processes: 3 1 4 2
Process Burst Time  Waiting Time    Turnaround Time
    1       4           0               4
    3       5           4               9
    0       8           9               17
    2       9           17              26
Average waiting time is: 7.50
Average Turnaround Time is: 14.00
```

Result:

The program implements the Priority Scheduling technique, calculating waiting time, turnaround time, and averages, and executed successfully.

Ex. No.: 6d)
Date: 26.03.2025

ROUND ROBIN SCHEDULING

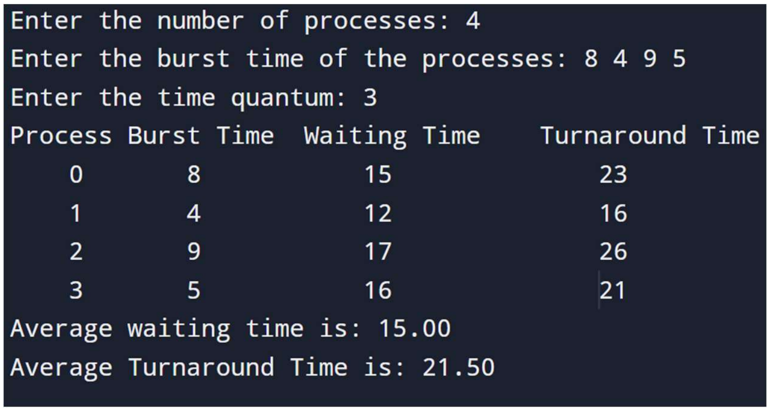
Aim:

To implement the Round Robin (RR) scheduling technique.

Program:

```
#include <stdio.h>
int main(){
    int n,i,tq;
    printf("Enter the number of processes: ");
    scanf("%d",&n);
    int bt[n],wt[n],tat[n],rem_bt[n],p[n];
    printf("Enter the burst time of the processes: ");
    for(i=0;i<n;i++){
        scanf("%d",&bt[i]);
        rem_bt[i]=bt[i];
        p[i]=i;
    }
    printf("Enter the time quantum: ");
    scanf("%d",&tq);
    int t=0,done;
    while(1){
        done=1;
        for(i=0;i<n;i++){
            if(rem_bt[i]>0){
                done=0;
                if(rem_bt[i]>tq){
                    t+=tq;
                    rem_bt[i]-=tq;
                }else{
                    t+=rem_bt[i];
                    wt[i]=t-bt[i];
                    rem_bt[i]=0;
                }
            }
        }
        if(done==1) break;
    }
    int total_wt=0,total_tat=0;
    for(i=0;i<n;i++){
        tat[i]=bt[i]+wt[i];
        total_wt+=wt[i];
        total_tat+=tat[i];
    }
    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for(i=0;i<n;i++) printf("%d\t%d\t\t%d\t\t%d\n",p[i],bt[i],wt[i],tat[i]);
    printf("Average waiting time is: %.2f\n",(float)total_wt/n);
    printf("Average Turnaround Time is: %.2f\n",(float)total_tat/n);
    return 0;
}
```

Output:



```
Enter the number of processes: 4
Enter the burst time of the processes: 8 4 9 5
Enter the time quantum: 3
Process Burst Time  Waiting Time    Turnaround Time
    0       8           15              23
    1       4           12              16
    2       9           17              26
    3       5           16              21
Average waiting time is: 15.00
Average Turnaround Time is: 21.50
```

Result:

The program implements the Round Robin Scheduling technique, calculates waiting time, turnaround time, averages, and executed successfully.

Ex. No.: 7
Date: 02.04.2025

IPC USING SHARED MEMORY

Aim:

To write a C program to do Inter Process Communication (IPC) using shared memory between sender process and receiver process.

Program:

sender.c

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <string.h>
#include <unistd.h>
int main(){
   key_t key=1234;
   int shmid=shmget(key,1024,0666|IPC_CREAT);
   char *str=(char*)shmat(shmid,(void*)0,0);
   sprintf(str,"Welcome to Shared Memory");
   sleep(5);
   shmdt(str);
   return 0;
}
```

receiver.c

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int main(){
   key_t key=1234;
   int shmid=shmget(key,1024,0666);
   char *str=(char*)shmat(shmid,(void*)0,0);
   printf("Message Received: %s\n",str);
   shmdt(str);
   return 0;
}
```

Output:

Terminal 1:

```
[root@localhost student]# gcc sender.c -o sender
[root@localhost student]# ./sender
```

Terminal 2:

```
[root@localhost student]# gcc receiver.c -o receiver
[root@localhost student]# ./receiver
Message Received: Welcome to Shared Memory
[root@localhost student]#
```

Result:

The program for Inter Process Communication using shared memory was executed successfully.

Ex. No.: 8
Date:16.04.2025

PRODUCER CONSUMER USING SEMAPHORES

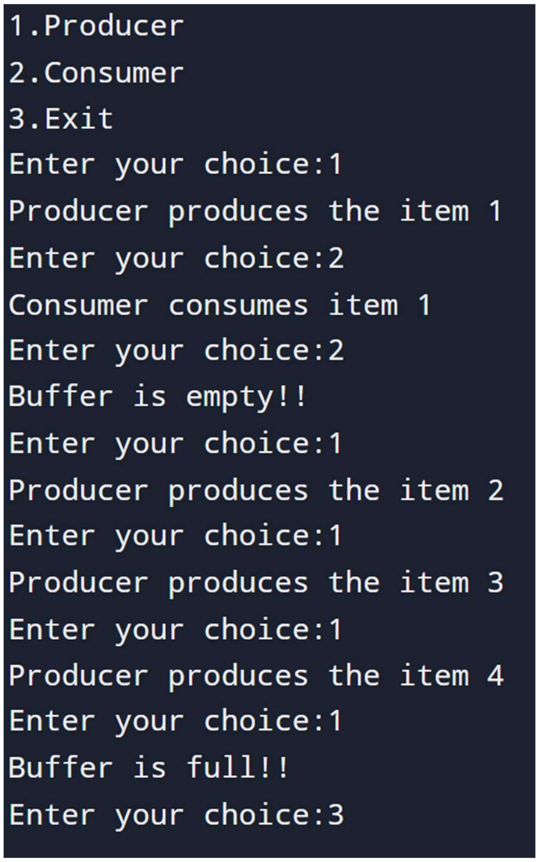
Aim:

To write a program to implement solution to producer consumer problem using semaphores.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#define SIZE 3
int buffer[SIZE], in=0, out=0, count=0, item=0;
sem_t empty, full, mutex;

void *producer(){
    if(count==SIZE){
        printf("Buffer is full!!\n");
        return NULL;
    }
    sem_wait(&empty);
    sem_wait(&mutex);
    item++;
    buffer[in]=item;
    in=(in+1)%SIZE;
    count++;
    printf("Producer produces the item %d\n", item);
    sem_post(&mutex);
    sem_post(&full);
    return NULL;
}

void *consumer(){
    if(count==0){
        printf("Buffer is empty!!\n");
        return NULL;
    }
    sem_wait(&full);
    sem_wait(&mutex);
    int data=buffer[out];
    out=(out+1)%SIZE;
    count--;
    printf("Consumer consumes item %d\n", data);
    sem_post(&mutex);
    sem_post(&empty);
    return NULL;
}

int main(){
    sem_init(&empty,0,SIZE);
    sem_init(&full,0,0);
    sem_init(&mutex,0,1);
    int choice;
    while(1){
        printf("1.Producer\n2.Consumer\n3.Exit\nEnter your choice:");
        scanf("%d",&choice);
        pthread_t t;
        if(choice==1)
            pthread_create(&t,NULL,producer,NULL);
        else if(choice==2)
            pthread_create(&t,NULL,consumer,NULL);
        else
            exit(0);
        pthread_join(t,NULL);
    }
    return 0;
}
```

Output:



```
1.Producer
2.Consumer
3.Exit
Enter your choice:1
Producer produces the item 1
Enter your choice:2
Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:1
Producer produces the item 2
Enter your choice:1
Producer produces the item 3
Enter your choice:1
Producer produces the item 4
Enter your choice:1
Buffer is full!!
Enter your choice:3
```

Result:

The program to solve the producer-consumer problem using semaphores was executed successfully.

Ex. No.: 9
Date: 17.04.2025

DEADLOCK AVOIDANCE

Aim:

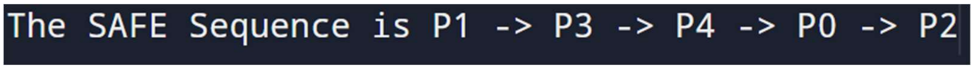
To find out a safe sequence using Banker's algorithm for deadlock avoidance.

Program:

```
#include <stdio.h>
#define P 5
#define R 3

int main(){
    int i, j, k;
    int alloc[P][R] = { {0, 1, 0}, {2, 0, 0}, {3, 0, 2}, {2, 1, 1}, {0, 0, 2} };
    int max[P][R] = { {7, 5, 3}, {3, 2, 2}, {9, 0, 2}, {2, 2, 2}, {4, 3, 3} };
    int avail[R] = {3, 3, 2};
    int f[P], ans[P], ind=0;
    for(k=0;k<P;k++) f[k]=0;
    int need[P][R];
    for(i=0;i<P;i++)
        for(j=0;j<R;j++)
            need[i][j]=max[i][j]-alloc[i][j];
    for(k=0;k<P;k++){
        for(i=0;i<P;i++){
            if(f[i]==0){
                int flag=0;
                for(j=0;j<R;j++){
                    if(need[i][j]>avail[j]){
                        flag=1;
                        break;
                    }
                }
                if(flag==0){
                    for(j=0;j<R;j++)
                        avail[j]+=alloc[i][j];
                    ans[ind++]=i;
                    f[i]=1;
                }
            }
        }
    }
    int flag=1;
    for(i=0;i<P;i++){
        if(f[i]==0){
            flag=0;
            printf("The system is not in a safe state\n");
            break;
        }
    }
    if(flag==1){
        printf("The SAFE Sequence is ");
        for(i=0;i<P-1;i++)
```

```
      printf("P%d -> ",ans[i]);
    printf("P%d\n",ans[P-1]);
  }
  return 0;
}
```

Output:



```
The SAFE Sequence is P1 -> P3 -> P4 -> P0 -> P2
```

Result:

The program to find the safe sequence using Banker’s Algorithm for deadlock avoidance was executed successfully.

Ex. No.: 10a)
Date: 19.04.2025

BEST FIT

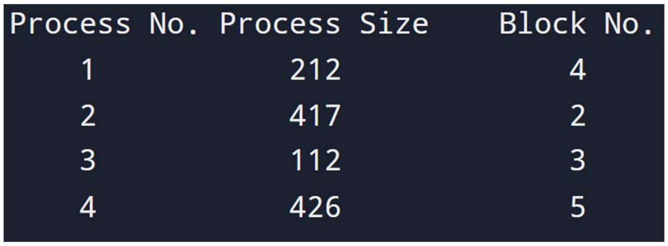
Aim:

To implement Best Fit memory allocation technique using Python.

Program:

```
def best_fit(block_size, process_size):
    n = len(block_size)
    m = len(process_size)
    allocation = [-1] * m

    for i in range(m):
        best_idx = -1
        for j in range(n):
            if block_size[j] >= process_size[i]:
                if best_idx == -1 or block_size[j] < block_size[best_idx]:
                    best_idx = j
        if best_idx != -1:
            allocation[i] = best_idx + 1
            block_size[best_idx] -= process_size[i]

    print("Process No.\tProcess Size\tBlock No.")
    for i in range(m):
        print(f"{i+1}\t\t{process_size[i]}\t\t", end="")
        if allocation[i] != -1:
            print(f"{allocation[i]}")
        else:
            print("Not Allocated")

# Sample input
block_size = [100, 500, 200, 300, 600]
process_size = [212, 417, 112, 426]

best_fit(block_size, process_size)
```

Output:



```
Process No. Process Size    Block No.
     1          212             4
     2          417             2
     3          112             3
     4          426             5
```

Result:

The program for Best Fit memory allocation technique was executed successfully and the output was verified.

Ex. No.: 10b)
Date: 19.04.2025

FIRST FIT

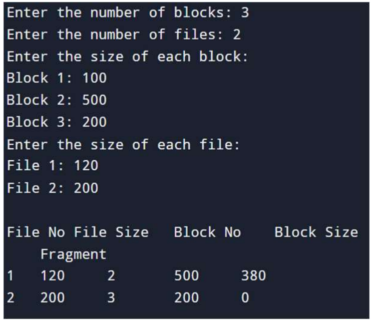
Aim:

To write a C program for implementation memory allocation methods for fixed partition using first fit.

Program:

```
#include <stdio.h>
#define max 25
int main() {
  int frag[max], b[max], f[max], i, j, nb, nf, temp;
  static int bf[max], ff[max];
  printf("Enter the number of blocks: ");
  scanf("%d", &nb);
  printf("Enter the number of files: ");
  scanf("%d", &nf);
  printf("Enter the size of the blocks:\n");
  for (i = 0; i < nb; i++)
    scanf("%d", &b[i]);
  printf("Enter the size of the files:\n");
  for (i = 0; i < nf; i++)
    scanf("%d", &f[i]);
  for (i = 0; i < nf; i++) {
    for (j = 0; j < nb; j++) {
      if (bf[j] != 1 && b[j] >= f[i]) {
        ff[i] = j;
        bf[j] = 1;
        frag[i] = b[j] - f[i];
        break;
      }
    }
  }
  printf("\nFile No\tFile Size\tBlock No\tBlock Size\tFragment\n");
  for (i = 0; i < nf; i++)
    printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", i + 1, f[i], ff[i] + 1, b[ff[i]], frag[i]);
  return 0;
}
```

Output:



```
Enter the number of blocks: 3
Enter the number of files: 2
Enter the size of each block:
Block 1: 100
Block 2: 500
Block 3: 200
Enter the size of each file:
File 1: 120
File 2: 200

File No File Size   Block No    Block Size
    Fragment
1   120     2       500     380
2   200     3       200     0
```

Result:

Thus, the program for First Fit memory allocation technique was executed successfully and the output was verified.

Ex. No.: 11a)

Date: 19.04.2025

FIFO PAGE REPLACEMENT

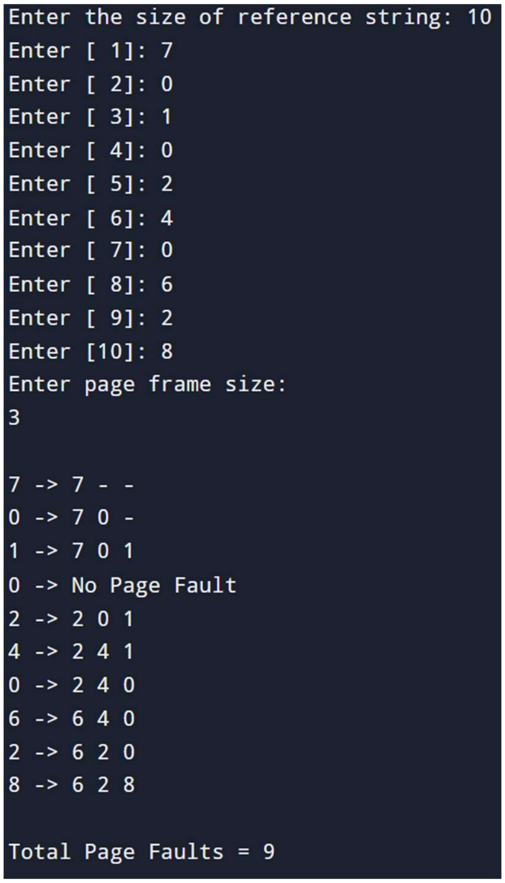
Aim:

To find out the number of page faults that occur using First-in First-out (FIFO) page replacement technique.

Program:

```
#include <stdio.h>
int main() {
    int referenceString[50], page[20], frames, refLen, i, j, k, avail, pageFaults = 0, next = 0;
    printf("Enter the size of reference string: ");
    scanf("%d", &refLen);

    for (i = 0; i < refLen; i++) {
        printf("Enter [%d] : ", i + 1);
        scanf("%d", &referenceString[i]);
    }
    printf("Enter page frame size: ");
    scanf("%d", &frames);

    for (i = 0; i < frames; i++)
        page[i] = -1;

    for (i = 0; i < refLen; i++) {
        avail = 0;
        for (j = 0; j < frames; j++) {
            if (page[j] == referenceString[i]) {
                avail = 1;
                break;
            }
        }
        if (avail == 0) {
            page[next] = referenceString[i];
            next = (next + 1) % frames;
            pageFaults++;

            for (k = 0; k < frames; k++)
                page[k] != -1 ? printf("%d ", page[k]) : printf("- ");
            printf("-> Page Fault\n");
        } else {
            for (k = 0; k < frames; k++)
                page[k] != -1 ? printf("%d ", page[k]) : printf("- ");
            printf("-> No Page Fault\n");
        }
    }
    printf("Total Page Faults: %d\n", pageFaults);
    return 0;
}
```

Output:



```
Enter the size of reference string: 10
Enter [ 1]: 7
Enter [ 2]: 0
Enter [ 3]: 1
Enter [ 4]: 0
Enter [ 5]: 2
Enter [ 6]: 4
Enter [ 7]: 0
Enter [ 8]: 6
Enter [ 9]: 2
Enter [10]: 8
Enter page frame size:
3

7 -> 7 - -
0 -> 7 0 -
1 -> 7 0 1
0 -> No Page Fault
2 -> 2 0 1
4 -> 2 4 1
0 -> 2 4 0
6 -> 6 4 0
2 -> 6 2 0
8 -> 6 2 8

Total Page Faults = 9
```

Result:

Thus, the program to implement FIFO Page Replacement was executed successfully and the number of page faults was determined correctly.

Ex. No.: 11b)
Date: 19.04.2025

LRU

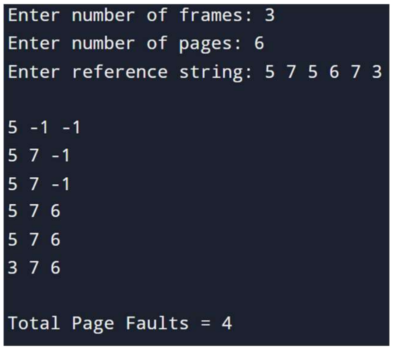
Aim:

To write a c program to implement LRU page replacement algorithm

Program:

```
#include <stdio.h>
int main() {
    int f[10], p[50], n, m, i, j, k, pos, pf = 0, lru[10], least;

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

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

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

    for (i = 0; i < n; i++) {
        f[i] = -1;
        lru[i] = 0;
    }

    printf("\n");

    for (i = 0; i < m; i++) {
        int found = 0;

        for (j = 0; j < n; j++) {
            if (f[j] == p[i]) {
                found = 1;
                lru[j] = i;
                break;
            }
        }

        if (!found) {
            if (pf < n) {
                f[pf] = p[i];
                lru[pf] = i;
            } else {
                least = lru[0];
                pos = 0;
                for (j = 1; j < n; j++) {
                    if (lru[j] < least) {
                        least = lru[j];
                        pos = j;
                    }
                }
                f[pos] = p[i];
                lru[pos] = i;
            }
            pf++;
        }

        for (k = 0; k < n; k++) {
            if (f[k] != -1)
                printf("%d ", f[k]);
            else
                printf("-1 ");
        }
        printf("\n");
    }

    printf("\nTotal Page Faults = %d\n", pf);
    return 0;
}
```

Output:



```
Enter number of frames: 3
Enter number of pages: 6
Enter reference string: 5 7 5 6 7 3

5 -1 -1
5 7 -1
5 7 -1
5 7 6
5 7 6
3 7 6

Total Page Faults = 4
```

Result:

Thus, the LRU Page Replacement Algorithm was successfully implemented, and the number of page faults was calculated based on the reference string.

Ex. No.: 11c)
Date: 19.04.2025

Optimal

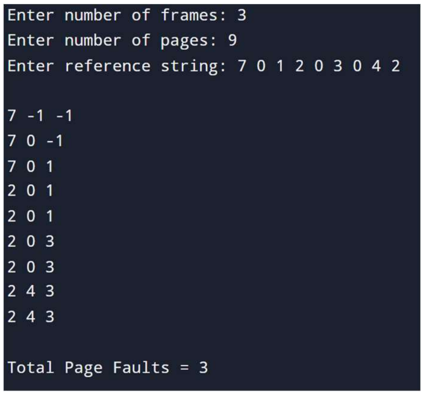
Aim:

To write a c program to implement Optimal page replacement.

Program:

```
#include <stdio.h>
int main() {
    int f[10], p[50], i, j, k, pos, pf = 0, n, m, found, farthest, index;

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

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

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

    for (i = 0; i < n; i++)
        f[i] = -1;

    printf("\n");

    for (i = 0; i < m; i++) {
        found = 0;

        for (j = 0; j < n; j++) {
            if (f[j] == p[i]) {
                found = 1;
                break;
            }
        }

        if (!found) {
            if (pf < n) {
                f[pf++] = p[i];
            } else {
                farthest = -1;
                index = -1;
                for (j = 0; j < n; j++) {
                    int next = -1;
                    for (k = i + 1; k < m; k++) {
                        if (f[j] == p[k]) {
                            next = k;
                            break;
                        }
                    }
                    if (next == -1) {
                        index = j;
                        break;
                    } else if (next > farthest) {
                        farthest = next;
                        index = j;
                    }
                }
                f[index] = p[i];
            }
        }

        for (j = 0; j < n; j++) {
            if (f[j] != -1)
                printf("%d ", f[j]);
            else
                printf("-1 ");
        }
        printf("\n");
    }

    printf("\nTotal Page Faults = %d\n", pf);
    return 0;
}
```

Output:



```
Enter number of frames: 3
Enter number of pages: 9
Enter reference string: 7 0 1 2 0 3 0 4 2

7 -1 -1
7 0 -1
7 0 1
2 0 1
2 0 1
2 0 3
2 0 3
2 4 3
2 4 3

Total Page Faults = 3
```

Result:

Thus, the Optimal Page Replacement Algorithm was successfully implemented, and the number of page faults was calculated based on the reference string.

Ex. No.: 12
Date: 19.04.2025

File Organization Technique- Single and Two level directory

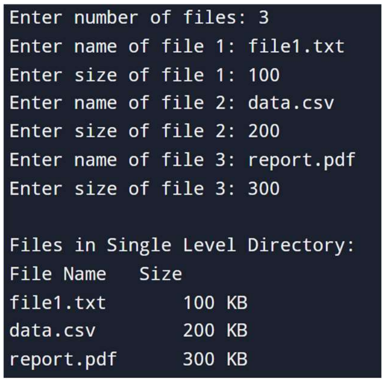
Aim:

To implement File Organization Structures in C are
a. Single Level Directory
b. Two-Level Directory

a. Single Level Directory
Program:

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

struct File {
    char name[20];
    int size;
};

int main() {
    struct File files[20];
    int n, i;

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

    for (i = 0; i < n; i++) {
        printf("Enter name of file %d: ", i + 1);
        scanf("%s", files[i].name);
        printf("Enter size of file %d: ", i + 1);
        scanf("%d", &files[i].size);
    }

    printf("\nFiles in Single Level Directory:\n");
    printf("File Name\tSize\n");
    for (i = 0; i < n; i++) {
        printf("%s\t\t%d KB\n", files[i].name, files[i].size);
    }

    return 0;
}
```

Output:

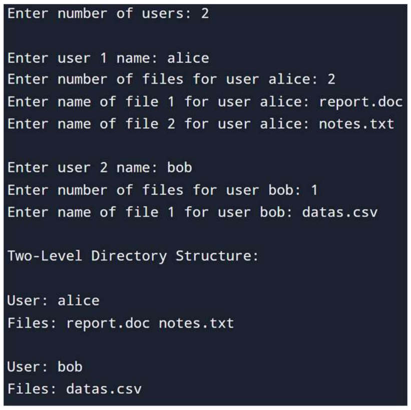


```
Enter number of files: 3
Enter name of file 1: file1.txt
Enter size of file 1: 100
Enter name of file 2: data.csv
Enter size of file 2: 200
Enter name of file 3: report.pdf
Enter size of file 3: 300

Files in Single Level Directory:
File Name   Size
file1.txt       100 KB
data.csv        200 KB
report.pdf      300 KB
```

Program:

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

struct File {
    char name[20];
};

struct Directory {
    char user[20];
    struct File files[10];
    int fileCount;
};

int main() {
    struct Directory dirs[10];
    int n, i, j;

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

    for (i = 0; i < n; i++) {
        printf("\nEnter user %d name: ", i + 1);
        scanf("%s", dirs[i].user);
        printf("Enter number of files for user %s: ", dirs[i].user);
        scanf("%d", &dirs[i].fileCount);

        for (j = 0; j < dirs[i].fileCount; j++) {
            printf("Enter name of file %d for user %s: ", j + 1, dirs[i].user);
            scanf("%s", dirs[i].files[j].name);
        }
    }

    printf("\nTwo-Level Directory Structure:\n");
    for (i = 0; i < n; i++) {
        printf("\nUser: %s\n", dirs[i].user);
        printf("Files: ");
        for (j = 0; j < dirs[i].fileCount; j++) {
            printf("%s ", dirs[i].files[j].name);
        }
        printf("\n");
    }

    return 0;
}
```

Output:



```
Enter number of users: 2

Enter user 1 name: alice
Enter number of files for user alice: 2
Enter name of file 1 for user alice: report.doc
Enter name of file 2 for user alice: notes.txt

Enter user 2 name: bob
Enter number of files for user bob: 1
Enter name of file 1 for user bob: datas.csv

Two-Level Directory Structure:

User: alice
Files: report.doc notes.txt

User: bob
Files: datas.csv
```

Result:

Thus, the Single level and Two level directory program was implemented successfully.