1. Change your password to a password you would like to use for the remainder of the semester.

kali@kali ~ $ passwd
Changing password for kali.
Old Password:
New Password:
Retype New Password:

2. display the systems date

kali@kali ~ $ date
Wed Jul 31 13:24:43 IST 2024

3.count the no. of lines in the /etc/passwd file.

kali@kali ~ $ wc -l /etc/passwd
57 /etc/passwd

4. find out who else is on the system.

kali@kali ~ $ whoami
kali

5. direct the output of the man pages for the date command to a file named mydate.

kali@kali ~ $ touch mydate
kali@kali ~ $ man date>mydate
kali@kali ~ $ cat mydate

DATE(1) BSD General Commands Manual DATE(1)

NAME

date -- display or set date and time

SYNOPSIS

date [**-jRu**] [**-r** seconds | filename] [**-v** [**+**|**-**]val[ymwdHMS]] ...
[**+**output_fmt]
date [**-jnu**] [[[mm]dd]HH]MM[[cc]yy][.ss]
date [**-jnRu**] **-f** input_fmt new_date [**+**output_fmt]
date [**-d** dst] [**-t** minutes_west]

DESCRIPTION

When invoked without arguments, the **date** utility displays the current date and time. Otherwise, depending on the options specified, **date** will set the date and time or print it in a user-defined way.

The **date** utility displays the date and time read from the kernel clock.

When used to set the date and time, both the kernel clock and the hardware clock are updated.

6. create a subdirectory called mydir
kali@kali ~ $ mkdir mydir

7. move the file mydate into the new subdirectory
kali@kali ~ $ mv mydate mydir
kali@kali ~ $ ls mydir
mydate

8. go to the subdirectory mydir and copy the file mydate to a new file called ourdate.
kali@kali ~/mydir $ cp ourdate.txt mydate.txt
kali@kali ~ /mydir $ more ourdate.txt

9. list the contents of mydir
kali@kali ~ $ ls mydir
mydate.txt ourdate.txt

10. do a long listing on the ile ourdate and note the permission
kali@kali ~ /mydir $ ls -l ourdate.txt
-rw-r--r-- 1 kali kali 0 Jul 31 07:26 ourdate.txt

11. display the name of the current directory starting from the root.
kali@kali ~ $ pwd
/home/kali

12. move the files inthe directory mydir back to your home directory
kali@kali ~ $ mv mydir kali
kali@kali ~ $ ls kali
mydate.txt

13. Display the first 5 lines of mydate
kali@kali ~ $ head -n 5 mydate.txt

DATE(1) BSD General Commands Manual DATE(1)

NAME
date -- display or set date and time

14. Display the last 8 lines of mydate
kali@kali ~ $ tail -n 8 mydate.txt
The date utility is expected to be compatible with IEEE Std 1003.2

(``POSIX.2''). The -d, -f, -j, -n, -r, -t, and -v options are all extensions to the standard.

HISTORY
A date command appeared in Version 1 AT&T UNIX.

BSD May 7, 2015 BSD

15. remove the directory mydir

```
kali@kali ~ $ rmdir mydir
rmdir: mydir: No such file or directory
```

16. redirect the output of the long listing of files to a file named list.

```
kali@kali ~ $ ls -l > list.txt
kali@kali ~ $ more list.txt
total 60
-rw-r--r-- 1 kali kali    0 Jul 31 01:36 '=18]'
drwxr-xr-x 2 kali kali 4096 Jul 23 13:24  Desktop
-rwxr--r-- 1 kali kali  114 Jul 31 02:22  divisors.sh
drwxr-xr-x 2 kali kali 4096 Feb  9 06:01  Documents
drwxr-xr-x 2 kali kali 4096 Feb  9 06:32  Downloads
-rwxr--r-- 1 kali kali  263 Jul 31 01:46  drvLic.sh
-rwxr--r-- 1 kali kali  306 Jul 31 02:13  gcd.sh
drwxr-xr-x 2 kali kali 4096 Jul 31 07:26  kali
drwxr-xr-x 2 kali kali 4096 Feb  9 06:01  Music
-rwxr--r-- 1 kali kali  532 Jul 31 01:55  parityCheck.sh
drwxr-xr-x 2 kali kali 4096 Feb  9 06:01  Pictures
drwxr-xr-x 2 kali kali 4096 Feb  9 06:01  Public
-rwxr--r-- 1 kali kali  733 Jul 31 02:17  stringRearrange.sh
-rwxr--r-- 1 kali kali   27 Jul 31 01:28  sumofN.sh
drwxr-xr-x 2 kali kali 4096 Feb  9 06:01  Templates
drwxr-xr-x 2 kali kali 4096 Feb  9 06:01  Videos
```

17. Select any 5 capitals of states in India and enter them in a file named capitals1. Choose 5 more capitals and enter them in a file named capitals2. Choose 5 more capitals and enter them in a file named capitals. concatenate all 3 files and redirect the output to a file named capitals

```
kali@kali ~ $ echo "New Delhi\nMumbai\nKolkata\nChennai\nBengaluru" >
capitals1.txt
kali@kali ~ $ echo "Hyderabad\nAhemdabad\nJaipur\nLucknow\nBhopal" >
capitals2.txt
```

```
kali@kali ~ $ echo
"Patna\nThiruvananthapuram\nBhubaneshwar\nRaipur\nChandigarh\n" >
capitals3.txt
kali@kali ~ $ cat capitals1.txt capitals2.txt capitals3.txt > capitals.txt
kali@kali ~ $ more capitals.txt
New Delhi
Mumbai
Kolkata
Chennai
Bengaluru
Hyderabad
Ahemdabad
Jaipur
Lucknow
Bhopal
Patna
Thiruvananthapuram
Bhubaneshwar
Raipur
Chandigarh
```

18.Concatenate the files capitals2 at the end of file capitals.

```
kali@kali ~ $ cat capitals.txt capitals2.txt
New Delhi
Mumbai
Kolkata
Chennai
Bengaluru
Hyderabad
Ahemdabad
Jaipur
Lucknow
Bhopal
Patna
Thiruvananthapuram
Bhubaneshwar
Raipur
Chandigarh
```

19. Give read and write permissions to all users for the file capitals.

```
kali@kali ~ $ ls -l capitals.txt
-rw-r--r-- 1 kali kali 121 Jul 31 07:40 capitals.txt
kali@kali ~ $ chmod u+rw capitals.txt
kali@kali ~ $ ls -l capitals.txt
```

-rw-rw-rw- 1 kali kali 121 Jul 31 07:40 capitals.txt

20.Give permission only to the owner of the file capitals.

kali@kali ~ $ chmod 400 capitals.txt
kali@kali ~ $ ls -l capitals.txt
-r-------- 1 kali kali 121 Jul 31 07:40 capitals.txt
kali@kali ~ $ echo "\nvren"> capitals.txt
zsh: permission denied: capitals.txt

21. Create an alias to concatenate the 3 files capitals1, capitals2, capitals3 and redirectthe output to a file named capitals Activate the alias and make it run

kali@kali ~ $ alias concate='cat capitals1.txt capitals2.txt capitals3.txt > capitals.txt'
kali@kali ~ $ concate
kali@kali ~ $ more capitals.txt
New Delhi
Mumbai
Kolkata
Chennai
Bengaluru
Hyderabad
Ahemdabad
Jaipur
Lucknow
Bhopal
Patna
Thiruvananthapuram
Bhubaneshwar
Raipur
Chandigarh

22. Find out the number of times “the” string appears in the file mydate

kali@kali ~ $ grep -wc "the" mydate.txt
96

23. Find out the line numbers on which the string “date” exists in mydate.

kali@kali ~ $ grep -n "date" mydate.txt
5: date -- display or set date and time
16: date and time. Otherwise, depending on the options specified, date will
17: set the date and time or print it in a user-defined way.
19: The date utility displays the date and time read from the kernel clock.
20: When used to set the date and time, both the kernel clock and the hard-
21: ware clock are updated.
23: Only the superuser may set the date, and if the system securelevel (see

37: -j Do not try to set the date. This allows you to use the -f flag
38: in addition to the + option to convert one date format to
46: -R Use RFC 2822 date and time output format. This is equivalent to
51: Print the date and time represented by seconds, where seconds is
57: Print the date and time of the last modification of filename.
64: -u Display or set the date in UTC (Coordinated Universal) time.
66: -v Adjust (i.e., take the current date and display the result of the
67: adjustment; not actually set the date) the second, minute, hour,
69: preceded with a plus or minus sign, the date is adjusted forwards
71: relevant part of the date is set. The date can be adjusted as
82: used to specify which part of the date is to be adjusted.
86: date will be put forwards (or backwards) to the next (previous)
87: date that matches the given week day or month. This will not
88: adjust the date, if the given week day or month is the same as
91: When a date is adjusted to a specific value or in units greater
94: So, assuming the current date is March 26, 0:30 and that the DST
96: using -v +1H will adjust the date to March 26, 2:30. Likewise,
97: if the date is October 29, 0:30 and the DST adjustment means that
101: When the date is adjusted to a specific value that does not actu-
103: Europe/London timezone), the date will be silently adjusted for-
105: the date is adjusted to a specific value that occurs twice (for
107: set so that the date matches the earlier of the two times.
109: It is not possible to adjust a date to an invalid absolute day,
114: Adjusting the date by months is inherently ambiguous because a
115: month is a unit of variable length depending on the current date.
116: This kind of date adjustment is applied in the most intuitive
120: For example, using -v +1m on May 31 will adjust the date to June
122: date adjusted to the last day of February. This approach is also
125: months may take you to a different date.
130: string which specifies the format in which to display the date and time.
138: value for setting the system's notion of the current date and time. The
139: canonical representation for setting the date and time is:
159: TZ The timezone to use when displaying dates. The normal format is
161: command ``TZ=America/Los_Angeles date'' displays the current time
168: The date utility exits 0 on success, 1 if unable to set the date, and 2
169: if able to set the local date, but unable to set it globally.
174: date "+DATE: %Y-%m-%d%nTIME: %H:%M:%S"
183: date -v1m -v+1y
193: date -v1d -v3m -v0y -v-1d
201: date -v3m -v30d -v0y -v-1m
203: because there is no such date as the 30th of February.
207: date -v1d -v+1m -v-1d -v-fri
217: date 0613162785
219: sets the date to ``June 13, 1985, 4:27 PM''.

221: date "+%m%d%H%M%Y.%S"
223: may be used on one machine to print out the date suitable for setting on
228: date 1432
230: sets the time to 2:32 PM, without modifying the date.
234: date -j -f "%a %b %d %T %Z %Y" "`date`" "+%s"
252: 0 The date was written successfully
253: 1 Unable to set the date
254: 2 Able to set the local date, but unable to set it globally

24. Print all lines of mydate except those that have the letter "i" in them.
kali@kali ~ $ grep -v 'i' mydate.txt

25. List the words of 4 letters from the file mydate.
kali@kali ~ $ grep -o '\b\w\{4\}\b' mydate.txt

26. List 5 states in north east India in afile mystates. list their corresponding capitals in a file mycapitals. use the paste command to join the 2 files
kali@kali ~ $ echo "Maharashtra\nKarnataka\nTamil Nadu\n West Bengal\nRajasthan" > mystates.txt
kali@kali ~ $ more mystates.txt
Maharashtra
Karnataka
Tamil Nadu
West Bengal
Rajasthan

kali@kali ~ $ echo " Mumbai\n Bengaluru\n Chennai\n Kolkata\n Jaipur">mycapitals .txt
kali@kali ~ $ more mycapitals.txt
Mumbai
Bengaluru
Chennai
Kolkata
Jaipur

kali@kali ~ $ paste mystates.txt mycapitals.txt
Maharashtra Mumbai
Karnataka Bengaluru
Tamil Nadu Chennai
West Bengal Kolkata
Rajasthan Jaipur

27. Use the cut command to print the 1 st and 3 rd columns of the /etc/passwd file for all students in this class.

```
kali@kali ~ $ cut -d ':' -f 1,3 /etc/passwd
root:0
daemon:1
bin:2
sys:3
sync:4
games:5
man:6
lp:7
mail:8
news:9
uucp:10
proxy:13
www-data:33
backup:34
list:38
irc:39
_apt:42
nobody:65534
systemd-network:998
systemd-timesync:992
messagebus:100
tss:101
strongswan:102
tcpdump:103
usbmux:104
sshd:105
dnsmasq:106
avahi:107
speech-dispatcher:108
pulse:109
lightdm:110
saned:111
polkitd:991
rtkit:112
colord:113
nm-openvpn:114
```

nm-openconnect:115
_galera:116
mysql:117
stunnel4:990
_rpc:118
geoclue:119
Debian-snmp:120
sslh:121
ntpsec:122
redsocks:123
rwhod:124
_gophish:125
iodine:126
miredo:127
statd:128
redis:129
postgres:130
mosquitto:131
inetsim:132
_gvm:133
kali:1000

28. Count the number of people logged in and also trap the users in a file using the tee command.

```
kali@kali ~ $ whoami | wc -l | tee file.txt
    1
kali@kali ~ $ more file.txt
    1
```

29. Convert the contents of mystates into uppercase.

```
kali@kali ~ $ tr '[:lower:]' '[:upper:]' < mystates.txt
```

30. create any 2 files & display the common values between them.

```
kali@kali ~ $ echo "abc" > file1.txt
kali@kali ~ $ echo "xyz" > file2.txt
kali@kali ~ $ comm file1.txt file2.txt
abc
    xyz
```

Name: Sumit Pujari

Class: CS-SY **Div: C**

Roll no: 68

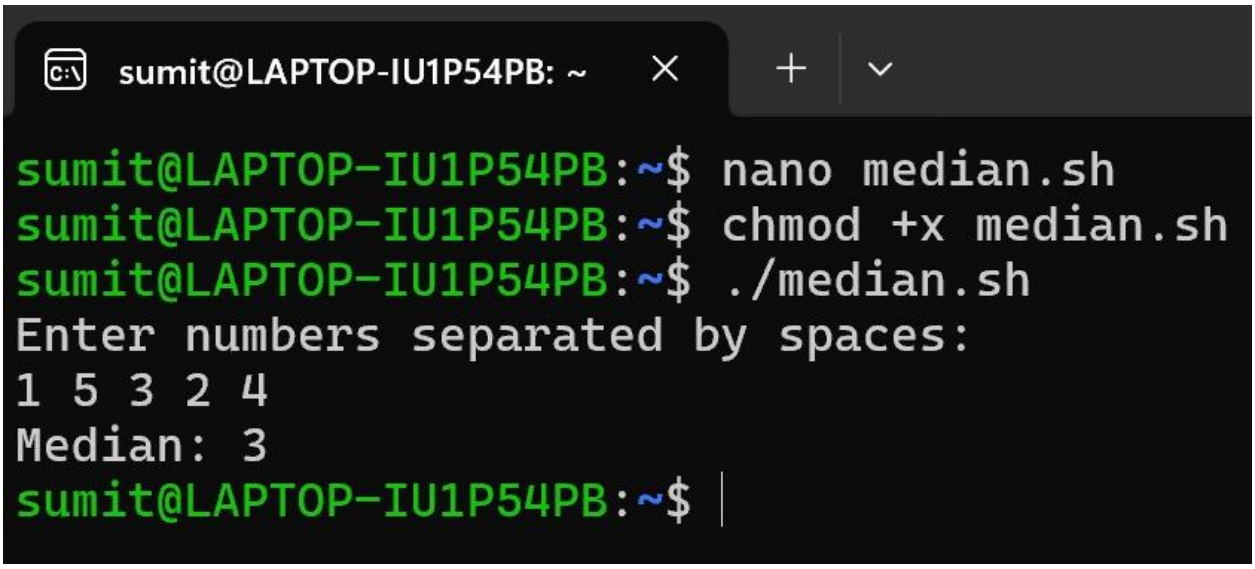
PRN: 12310079

Shell Scripting

1) **Array: Print median of list of numbers.**

```
#!/bin/bash

echo "Enter numbers separated by spaces:"
read -a numbers

sorted=($(echo "${numbers[@]}" | tr ' ' '\n' | sort -n))

length=${#sorted[@]}

if (( $length % 2 == 0 )); then
   mid=$((length / 2))
   median=$(( (sorted[mid-1] + sorted[mid]) / 2 ))
else
   mid=$((length / 2))
   median=${sorted[$mid]}
fi
echo "Median: $median"
```

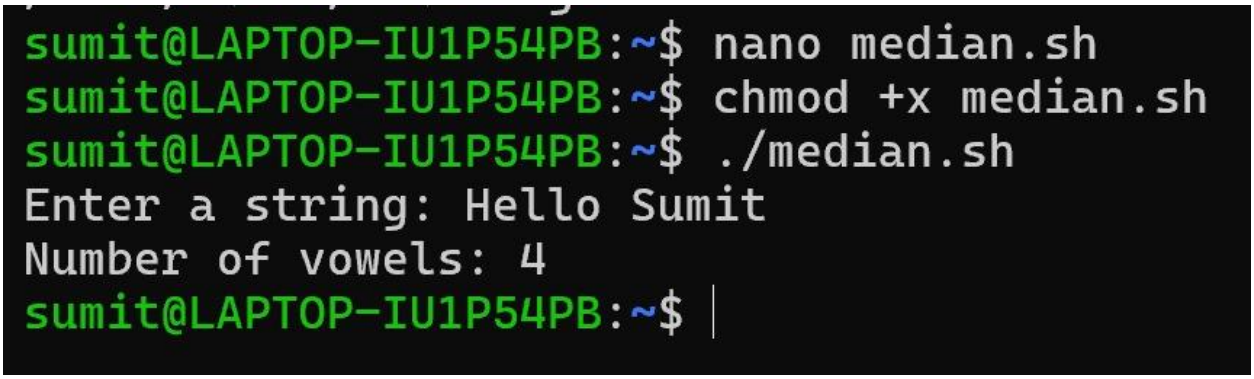
Output:



2) **String: Count the Number of Vowels in a String.**

```
#!/bin/bash
read -p "Enter a string: " str
vowel_count=$(echo $str | grep -o -i "[aeiou]" | wc -l)
echo "Number of vowels: $vowel_count"
```

Output:



3) Function: Convert given input decimal number to binary number

```
#!/bin/bash
decimal_to_binary() {
    local decimal=$1
    binary=""

    while [ $decimal -ne 0 ]; do
        remainder=$((decimal % 2))
        binary="${remainder}${binary}"
        decimal=$((decimal / 2))
    done
echo $binary
}

echo "Enter a decimal number:"
read decimal

binary=$(decimal_to_binary $decimal)
echo "Binary: $binary"
```

Output:

```
sumit@LAPTOP-IU1P54PB:~$ nano median.sh
sumit@LAPTOP-IU1P54PB:~$ sudo apt-get install bc
[sudo] password for sumit:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  bc
0 upgraded, 1 newly installed, 0 to remove and 127 not upgraded.
Need to get 87.6 kB of archives.
After this operation, 220 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu jammy/main amd64 bc amd64 1.07.1-3build1 [87.6 kB]
Fetched 87.6 kB in 1s (64.9 kB/s)
Selecting previously unselected package bc.
(Reading database ... 28546 files and directories currently installed.)
Preparing to unpack .../bc_1.07.1-3build1_amd64.deb ...
Unpacking bc (1.07.1-3build1) ...
Setting up bc (1.07.1-3build1) ...
Processing triggers for install-info (6.8-4build1) ...
Processing triggers for man-db (2.10.2-1) ...
sumit@LAPTOP-IU1P54PB:~$ chmod +x median.sh
sumit@LAPTOP-IU1P54PB:~$ ./median.sh
Enter a decimal number:
10
Binary: 1010
sumit@LAPTOP-IU1P54PB:~$
```

4) Arithmetic: Calculate the sum of square of first n natural numbers

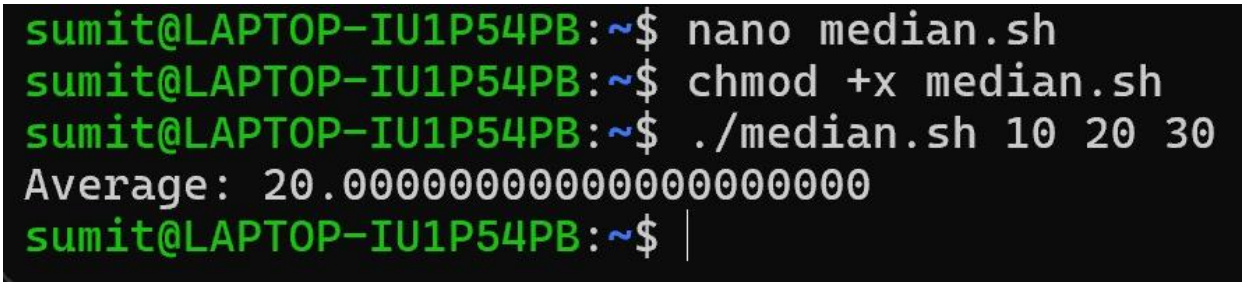
```
#!/bin/bash
# Sum of squares of first n natural numbers

read -p "Enter the value of n: " n
sum=0

for (( i=1; i<=n; i++ ))
do
  sum=$(( sum + i*i ))
done

echo "Sum of squares of first $n natural numbers: $sum"
```

```
sumit@LAPTOP-IU1P54PB:~$ nano median.sh
sumit@LAPTOP-IU1P54PB:~$
sumit@LAPTOP-IU1P54PB:~$ chmod +x median.sh
sumit@LAPTOP-IU1P54PB:~$ ./median.sh
Enter the value of n: 5
Sum of squares of first 5 natural numbers: 55
```

5) **Command line :-List of integers as command line arguments and calculates their average.**

```
#!/bin/bash

# Check if arguments are passed
if [ $# -eq 0 ]; then
    echo "No arguments provided. Please pass integers as arguments."
    exit 1
fi

# Calculate the sum of the integers
sum=0
for num in "$@"; do
    sum=$((sum + num))
done

# Calculate the average
average=$(echo "$sum / $#" | bc -l)
echo "Average: $average"
```

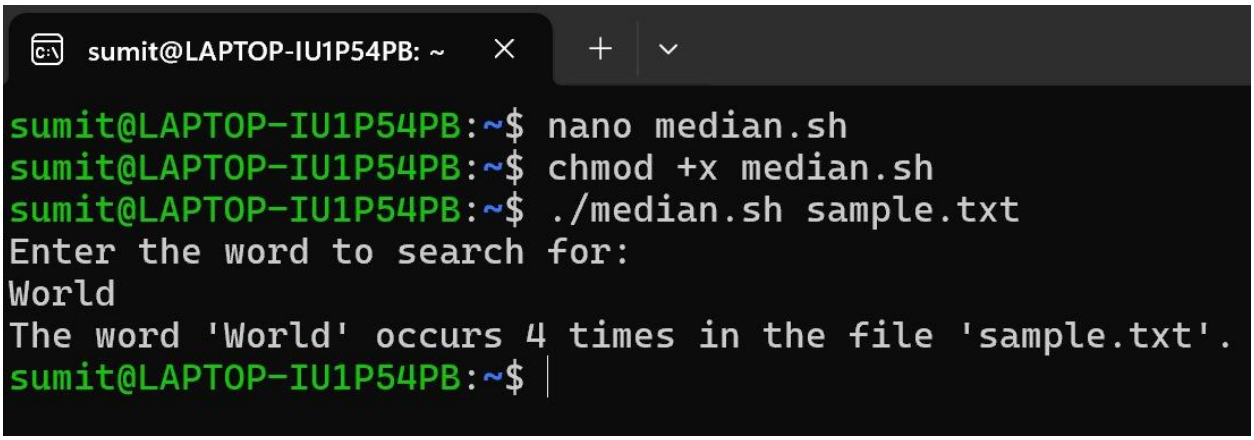
Output:



6) **Executing a linux command in shell :- Take a file name as an argument and counts the number of occurrences of a specific word provided by the user.**

```
#!/bin/bash

# Input: File name and word to search for
if [ $# -ne 1 ]; then
    echo "Usage: $0 <filename>"
    exit 1
fi

echo "Enter the word to search for:"
read word

# Count occurrences of the word
count=$(grep -o -i "$word" "$1" | wc -l)

echo "The word '$word' occurs $count times in the file '$1'."
```

Output:


sumit@LAPTOP-IU1P54PB: ~
sumit@LAPTOP-IU1P54PB:~$ nano median.sh
sumit@LAPTOP-IU1P54PB:~$ chmod +x median.sh
sumit@LAPTOP-IU1P54PB:~$ ./median.sh sample.txt
Enter the word to search for:
World
The word 'World' occurs 4 times in the file 'sample.txt'.
sumit@LAPTOP-IU1P54PB:~$

Name: Sumit Pujari

Class: CS-SY Div: C

Roll No: 68

PRN: 12310079

1) Producer-Consumer Problem

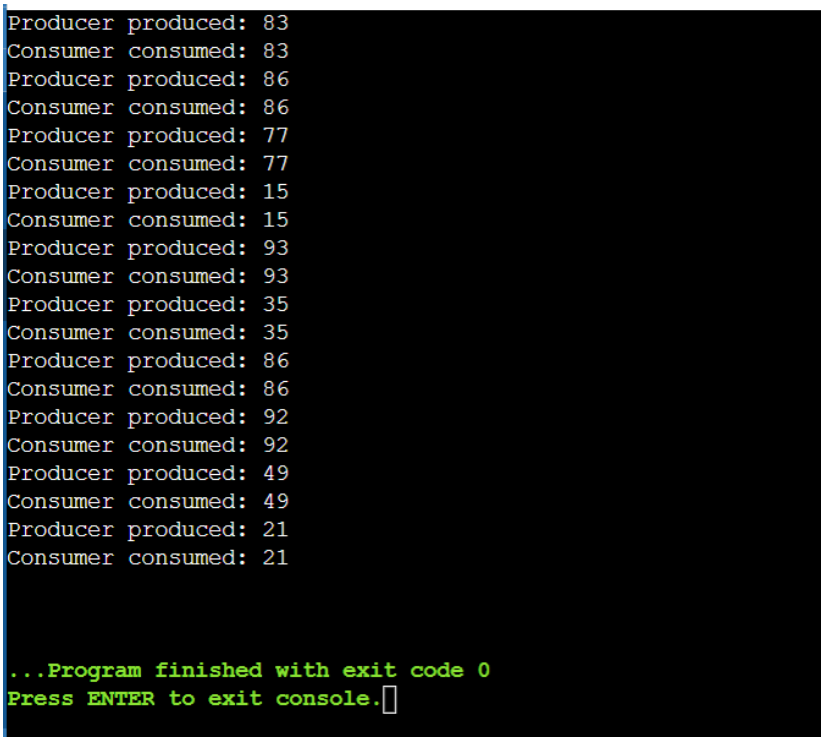
Code:

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

 #define BUFFER_SIZE 100

int buffer[BUFFER_SIZE];
int in = 0, out = 0;
sem_t empty, full;
pthread_mutex_t mutex;

void *producer(void *arg) {
    while (1) {
        int item = rand() % 100;
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);

        buffer[in] = item;
```

```
        printf("Producer produced: %d\n", item);
        in = (in + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex);
        sem_post(&full);
        usleep(500000);
    }
}

void *consumer(void *arg) {
    while (1) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);

        int item = buffer[out];
        printf("Consumer consumed: %d\n", item);
        out = (out + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
        usleep(500000);
    }
}

int main() {
    pthread_t prod_thread, cons_thread;
```

```
    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    pthread_mutex_init(&mutex, NULL);

    pthread_create(&prod_thread, NULL, producer, NULL);
    pthread_create(&cons_thread, NULL, consumer, NULL);

    getchar();

    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

Output:



2) Reader-Writers Problem

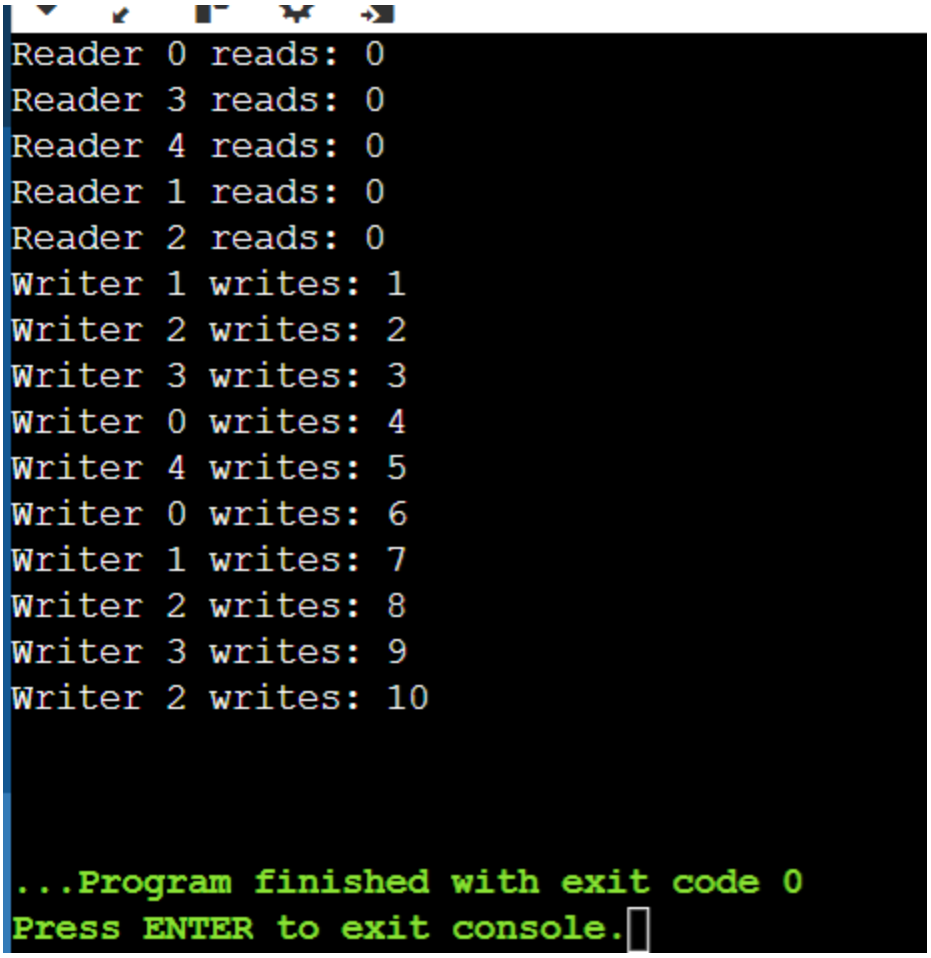
Code:

```
#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>


pthread_mutex_t mutex;

sem_t wrt;

int read_count = 0;

int shared_data = 0;


void *reader(void *arg) {
```

```
    while (1) {
        pthread_mutex_lock(&mutex);
        read_count++;
        if (read_count == 1) {
            sem_wait(&wrt);
        }
        pthread_mutex_unlock(&mutex);

        printf("Reader %ld reads: %d\n", (long)arg, shared_data);
        usleep(500000);

        pthread_mutex_lock(&mutex);
        read_count--;
        if (read_count == 0) {
            sem_post(&wrt);
        }
        pthread_mutex_unlock(&mutex);

        usleep(500000);
    }
}

void *writer(void *arg) {
    while (1) {
        sem_wait(&wrt);

        shared_data++;
```

```
        printf("Writer %ld writes: %d\n", (long)arg, shared_data);
        usleep(500000);

        sem_post(&wrt);
        usleep(500000);
    }
}

int main() {
    pthread_t readers[5], writers[5];

    pthread_mutex_init(&mutex, NULL);
    sem_init(&wrt, 0, 1);

    for (long i = 0; i < 5; i++) {
        pthread_create(&readers[i], NULL, reader, (void *)i);
        pthread_create(&writers[i], NULL, writer, (void *)i);
    }

    getchar();

    sem_destroy(&wrt);
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

Output:



3) Dining Philosophers Problem

Code:

```
#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

 #define N 5


sem_t forks[N];
```

```
pthread_t philosophers[N];

void *philosopher(void *num) {
    long id = (long)num;
    while (1) {
        printf("Philosopher %ld is thinking.\n", id);
        usleep(500000);

        sem_wait(&forks[id]);
        sem_wait(&forks[(id + 1) % N]);

        printf("Philosopher %ld is eating.\n", id);
        usleep(500000);

        sem_post(&forks[id]);
        sem_post(&forks[(id + 1) % N]);
    }
}

int main() {
    for (int i = 0; i < N; i++) {
        sem_init(&forks[i], 0, 1);
    }

    for (long i = 0; i < N; i++) {
        pthread_create(&philosophers[i], NULL, philosopher, (void *)i);
    }
```

```
    getchar();

    for (int i = 0; i < N; i++) {
        sem_destroy(&forks[i]);
    }

    return 0;
}
```

Output:

```
Philosopher 2 is thinking.
Philosopher 1 is thinking.
Philosopher 4 is thinking.
Philosopher 0 is thinking.
Philosopher 3 is thinking.
Philosopher 2 is eating.
Philosopher 4 is eating.
Philosopher 2 is thinking.
Philosopher 1 is eating.
Philosopher 3 is eating.
Philosopher 4 is thinking.
Philosopher 3 is thinking.
Philosopher 2 is eating.
Philosopher 1 is thinking.
Philosopher 0 is eating.
Philosopher 2 is thinking.
Philosopher 1 is eating.
Philosopher 4 is eating.
Philosopher 0 is thinking.
Philosopher 1 is thinking.
```

CPU Scheduling:

FCFS:

```
#include <stdio.h>
typedef struct {
   int id;   // Process ID
   int at;   // Arrival Time
   int bt;   // Burst Time
   int ct;   // Completion Time
   int tat;  // Turnaround Time
   int wt;   // Waiting Time
} Process;

void calculateTimes(Process p[], int n) {
   int currentTime = 0;

   for (int i = 0; i < n; i++) {
      if (currentTime < p[i].at) {
         currentTime = p[i].at;  // If CPU is idle
      }
      p[i].ct = currentTime + p[i].bt;  // Completion Time
      currentTime = p[i].ct;
      p[i].tat = p[i].ct - p[i].at;     // Turnaround Time
      p[i].wt = p[i].tat - p[i].bt;     // Waiting Time
   }
}

void printGanttChart(Process p[], int n) {
   printf("\nGantt Chart:\n");

   // Top line
   printf(" ");
```

```
    for (int i = 0; i < n; i++) {
        printf("-------");
    }
    printf("\n");

    // Process line
    printf("|");
    for (int i = 0; i < n; i++) {
        printf(" P%-3d |", p[i].id);
    }
    printf("\n");

    // Bottom line
    printf(" ");
    for (int i = 0; i < n; i++) {
        printf("-------");
    }
    printf("\n");

    // Time line
    printf("0");
    for (int i = 0; i < n; i++) {
        printf("      %d", p[i].ct);
    }
    printf("\n");
}

void printTable(Process p[], int n) {
    printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\t%d\t%d\n",
```

```
            p[i].id, p[i].at, p[i].bt, p[i].ct, p[i].tat, p[i].wt);
    }
}

int main() {
    int n;

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

    Process p[n];

    printf("Enter Arrival Time and Burst Time for each process:\n");
    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;  // Process ID
        printf("Process %d:\n", i + 1);
        printf("Arrival Time: ");
        scanf("%d", &p[i].at);
        printf("Burst Time: ");
        scanf("%d", &p[i].bt);
    }

    // Sort processes by arrival time (FCFS rule)
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (p[i].at > p[j].at) {
                Process temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
```

```
    }

    calculateTimes(p, n);
    printTable(p, n);
    printGanttChart(p, n);

    return 0;
}
```

```
Enter the number of processes: 4
Enter Arrival Time and Burst Time for each process:
Process 1:
Arrival Time: 0
Burst Time: 2
Process 2:
Arrival Time: 1
Burst Time: 2
Process 3:
Arrival Time: 5
Burst Time: 3
Process 4:
Arrival Time: 6
Burst Time: 4
```

SJF Non preemptive:

```
#include <stdio.h>

typedef struct {
    int id;   // Process ID
    int at;   // Arrival Time
    int bt;   // Burst Time
    int ct;   // Completion Time
    int tat;  // Turnaround Time
    int wt;   // Waiting Time
    int completed; // Flag to check if process is completed
} Process;

void calculateTimes(Process p[], int n) {
    int currentTime = 0, completed = 0;
    int ganttOrder[n], goIdx = 0;
```

```
while (completed < n) {
    int min_bt = 1e9, idx = -1;

    // Find the process with the shortest burst time that has arrived
    for (int i = 0; i < n; i++) {
        if (p[i].at <= currentTime && !p[i].completed && p[i].bt < min_bt) {
            min_bt = p[i].bt;
            idx = i;
        }
    }

    if (idx != -1) {
        // Process the selected job
        ganttOrder[goIdx++] = p[idx].id; // Add to Gantt order
        currentTime += p[idx].bt;
        p[idx].ct = currentTime;
        p[idx].tat = p[idx].ct - p[idx].at;
        p[idx].wt = p[idx].tat - p[idx].bt;
        p[idx].completed = 1; // Mark process as completed
        completed++;
    } else {
        // If no process has arrived yet, increment time
        currentTime++;
    }
}

// Print the Gantt Chart
printf("\nGantt Chart:\n");
printf(" ");
for (int i = 0; i < goIdx; i++) {
```

```
        printf("-------");
    }
    printf("\n|");
    for (int i = 0; i < goIdx; i++) {
        printf(" P%-3d |", ganttOrder[i]);
    }
    printf("\n ");
    for (int i = 0; i < goIdx; i++) {
        printf("-------");
    }
    printf("\n0");
    currentTime = 0;
    for (int i = 0; i < goIdx; i++) {
        for (int j = 0; j < n; j++) {
            if (p[j].id == ganttOrder[i]) {
                currentTime += p[j].bt;
                printf("     %d", currentTime);
                break;
            }
        }
    }
    printf("\n");
}

void printTable(Process p[], int n) {
    printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\t%d\t%d\n",
               p[i].id, p[i].at, p[i].bt, p[i].ct, p[i].tat, p[i].wt);
    }
}
```

```
int main() {
    int n;

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

    Process p[n];

    printf("Enter Arrival Time and Burst Time for each process:\n");
    for (int i = 0; i < n; i++) {
        p[i].id = i + 1; // Process ID
        p[i].completed = 0; // Initially, no process is completed
        printf("Process %d:\n", i + 1);
        printf("Arrival Time: ");
        scanf("%d", &p[i].at);
        printf("Burst Time: ");
        scanf("%d", &p[i].bt);
    }

    calculateTimes(p, n);
    printTable(p, n);

    return 0;
}
```

```
Enter the number of processes: 4
Enter Arrival Time and Burst Time for each process:
Process 1:
Arrival Time: 1
Burst Time: 3
Process 2:
Arrival Time: 2
Burst Time: 4
Process 3:
Arrival Time: 1
Burst Time: 2
Process 4:
Arrival Time: 4
Burst Time: 4
```

SJF preemptive:

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

typedef struct {
    int id;    // Process ID
    int at;    // Arrival Time
    int bt;    // Burst Time
    int ct;    // Completion Time
    int tat;   // Turnaround Time
    int wt;    // Waiting Time
    int rt;    // Remaining Time
} Process;

void calculateTimes(Process p[], int n) {
    int currentTime = 0, completed = 0, min_rt, idx = -1;
    int ganttChart[200], timeStamps[200], gcIdx = 0;

    // Initialize remaining time for all processes
    for (int i = 0; i < n; i++) {
        p[i].rt = p[i].bt;
    }

    while (completed < n) {
        min_rt = INT_MAX;
        idx = -1;

        // Find process with the shortest remaining time that has arrived
        for (int i = 0; i < n; i++) {
            if (p[i].at <= currentTime && p[i].rt > 0 && p[i].rt < min_rt) {
                min_rt = p[i].rt;
```

```
            idx = i;
        }
    }

    if (idx != -1) {
        // Process the selected job for 1 unit of time
        ganttChart[gcIdx] = p[idx].id;
        timeStamps[gcIdx] = currentTime;
        gcIdx++;
        currentTime++;
        p[idx].rt--;

        // If process is completed
        if (p[idx].rt == 0) {
            p[idx].ct = currentTime;
            p[idx].tat = p[idx].ct - p[idx].at;
            p[idx].wt = p[idx].tat - p[idx].bt;
            completed++;
        }
    } else {
        // If no process is ready to execute, increment time
        ganttChart[gcIdx] = -1; // Idle state
        timeStamps[gcIdx] = currentTime;
        gcIdx++;
        currentTime++;
    }
}
timeStamps[gcIdx] = currentTime; // Final timestamp

// Print Gantt Chart
printf("\nGantt Chart:\n");
```

```
    for (int i = 0; i < gcIdx; i++) {
        if (i == 0 || ganttChart[i] != ganttChart[i - 1]) {
            printf(" ------");
        }
    }
    printf("\n|");
    for (int i = 0; i < gcIdx; i++) {
        if (i == 0 || ganttChart[i] != ganttChart[i - 1]) {
            if (ganttChart[i] == -1) {
                printf(" IDLE |");
            } else {
                printf(" P%-3d |", ganttChart[i]);
            }
        }
    }
    printf("\n");
    for (int i = 0; i < gcIdx; i++) {
        if (i == 0 || ganttChart[i] != ganttChart[i - 1]) {
            printf(" ------");
        }
    }
    printf("\n%d", timeStamps[0]);
    for (int i = 1; i <= gcIdx; i++) {
        if (i == gcIdx || ganttChart[i] != ganttChart[i - 1]) {
            printf("      %d", timeStamps[i]);
        }
    }
    printf("\n");
}

void printTable(Process p[], int n) {
```

```
    printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\t%d\t%d\n",
                p[i].id, p[i].at, p[i].bt, p[i].ct, p[i].tat, p[i].wt);
    }
}

int main() {
    int n;

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

    Process p[n];

    printf("Enter Arrival Time and Burst Time for each process:\n");
    for (int i = 0; i < n; i++) {
        p[i].id = i + 1; // Process ID
        printf("Process %d:\n", i + 1);
        printf("Arrival Time: ");
        scanf("%d", &p[i].at);
        printf("Burst Time: ");
        scanf("%d", &p[i].bt);
    }

    calculateTimes(p, n);
    printTable(p, n);

    return 0;
}
```

```
Enter the number of processes: 4
Enter Arrival Time and Burst Time for each process:
Process 1:
Arrival Time: 0
Burst Time: 5
Process 2:
Arrival Time: 1
Burst Time: 3
Process 3:
Arrival Time: 2
Burst Time: 4
Process 4:
Arrival Time: 4
Burst Time: 1
```

Round robin:

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

typedef struct {
    int id;    // Process ID
    int at;    // Arrival Time
    int bt;    // Burst Time
    int ct;    // Completion Time
    int tat;   // Turnaround Time
    int wt;    // Waiting Time
    int rt;    // Remaining Time
} Process;

void calculateTimes(Process p[], int n, int tq) {
    int currentTime = 0, completed = 0;
    int ganttChart[200], timeStamps[200], gcIdx = 0;
    bool inQueue[100] = {false}; // To track if a process is in the ready queue

    // Initialize remaining times
    for (int i = 0; i < n; i++) {
```

```
    p[i].rt = p[i].bt;
}

// Queue simulation
int queue[100], front = 0, rear = 0;
queue[rear++] = 0; // Add the first process (assumes sorted by AT)
inQueue[0] = true;

while (completed < n) {
    int idx = queue[front++];
    if (front == 100) front = 0; // Circular queue

    // Add to Gantt Chart
    ganttChart[gcIdx] = p[idx].id;
    timeStamps[gcIdx++] = currentTime;

    // Process execution
    if (p[idx].rt <= tq) {
        currentTime += p[idx].rt;
        p[idx].rt = 0;
        p[idx].ct = currentTime;
        p[idx].tat = p[idx].ct - p[idx].at;
        p[idx].wt = p[idx].tat - p[idx].bt;
        completed++;
    } else {
        currentTime += tq;
        p[idx].rt -= tq;
    }

    // Check for newly arrived processes
    for (int i = 0; i < n; i++) {
```

```
        if (!inQueue[i] && p[i].at <= currentTime && p[i].rt > 0) {
            queue[rear++] = i;
            if (rear == 100) rear = 0; // Circular queue
            inQueue[i] = true;
        }
    }

    // Re-add the current process to the queue if it is not completed
    if (p[idx].rt > 0) {
        queue[rear++] = idx;
        if (rear == 100) rear = 0; // Circular queue
    }

    // If the queue becomes empty and there are remaining processes, advance time
    if (front == rear) {
        for (int i = 0; i < n; i++) {
            if (p[i].rt > 0) {
                queue[rear++] = i;
                if (rear == 100) rear = 0; // Circular queue
                inQueue[i] = true;
                break;
            }
        }
    }
}

// Final timestamp for Gantt Chart
timeStamps[gcIdx] = currentTime;

// Print Gantt Chart
printf("\nGantt Chart:\n");
```

```
    for (int i = 0; i < gcIdx; i++) {
        printf(" ------");
    }
    printf("\n|");
    for (int i = 0; i < gcIdx; i++) {
        printf(" P%-3d |", ganttChart[i]);
    }
    printf("\n");
    for (int i = 0; i < gcIdx; i++) {
        printf(" ------");
    }
    printf("\n%d", timeStamps[0]);
    for (int i = 1; i <= gcIdx; i++) {
        printf("    %d", timeStamps[i]);
    }
    printf("\n");
}

void printTable(Process p[], int n) {
    printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\t%d\t%d\n",
            p[i].id, p[i].at, p[i].bt, p[i].ct, p[i].tat, p[i].wt);
    }
}

int main() {
    int n, tq;

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

```
    Process p[n];

    printf("Enter Arrival Time and Burst Time for each process:\n");
    for (int i = 0; i < n; i++) {
        p[i].id = i + 1; // Process ID
        printf("Process %d:\n", i + 1);
        printf("Arrival Time: ");
        scanf("%d", &p[i].at);
        printf("Burst Time: ");
        scanf("%d", &p[i].bt);
    }

    printf("Enter the Time Quantum: ");
    scanf("%d", &tq);

    calculateTimes(p, n, tq);
    printTable(p, n);

    return 0;
}
```

```
Enter the number of processes: 4
Enter Arrival Time and Burst Time for each process:
Process 1:
Arrival Time: 0
Burst Time: 5
Process 2:
Arrival Time: 1
Burst Time: 4
Process 3:
Arrival Time: 2
Burst Time: 2
Process 4:
Arrival Time: 4
Burst Time: 1
Enter the Time Quantum: 2
```

Priority preemptive:

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

struct Process {
   int id;     // Process ID
   int at;     // Arrival Time
   int bt;     // Burst Time
   int orig_bt; // Original Burst Time to calculate Waiting Time and Turnaround Time
   int pt;     // Priority (1 is highest)
   int ct;     // Completion Time
   int tat;    // Turnaround Time
   int wt;     // Waiting Time
};

int compare(const void *a, const void *b) {
   return ((struct Process *)a)->at - ((struct Process *)b)->at;
}

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

   struct Process *processes = (struct Process *)malloc(n * sizeof(struct Process));

   for (int i = 0; i < n; i++) {
      processes[i].id = i + 1; // Process ID starts from 1
      printf("Enter arrival time, burst time, and priority for process %d: ", i + 1);
      scanf("%d %d %d", &processes[i].at, &processes[i].bt, &processes[i].pt);
```

```
    processes[i].orig_bt = processes[i].bt; // Store the original burst time
  }

  // Sort processes based on arrival time
  qsort(processes, n, sizeof(struct Process), compare);

  int currentTime = 0;
  int completed = 0;
  int isCompleted[n];
  int currentProcess = -1;

  // Initialize completed array
  for (int i = 0; i < n; i++) {
    isCompleted[i] = 0;
  }

  int gantt[100]; // Store process IDs for Gantt chart
  int gantt_time[100]; // Store time for each segment
  int gantt_index = 0;

  while (completed < n) {
    int idx = -1;
    int highestPriority = INT_MAX;

    // Find the process with the highest priority that has arrived
    for (int i = 0; i < n; i++) {
      if (processes[i].at <= currentTime && !isCompleted[i] && processes[i].bt > 0 &&
processes[i].pt < highestPriority) {
        highestPriority = processes[i].pt;
        idx = i;
      }
```

```
    }

    if (idx != -1) {
        // Log the current process ID for Gantt chart
        if (currentProcess != idx) {
            if (currentProcess != -1) {
                gantt[gantt_index] = processes[currentProcess].id;
                gantt_time[gantt_index] = 1; // Add one time unit for the previous process
                gantt_index++;
            }
            currentProcess = idx;
        }

        processes[currentProcess].bt--; // Decrease burst time
        currentTime++;

        // Add time to Gantt chart for the current process
        if (gantt_index > 0 && gantt[gantt_index - 1] == processes[currentProcess].id) {
            gantt_time[gantt_index - 1]++;
        } else {
            gantt[gantt_index] = processes[currentProcess].id;
            gantt_time[gantt_index] = 1;
            gantt_index++;
        }

        if (processes[currentProcess].bt == 0) {
            processes[currentProcess].ct = currentTime;
            processes[currentProcess].tat = processes[currentProcess].ct - processes[currentProcess].at;
            processes[currentProcess].wt = processes[currentProcess].tat -
processes[currentProcess].orig_bt; // Use original burst time for wt calculation
            isCompleted[currentProcess] = 1; // Mark process as completed
```

```
            completed++;
            currentProcess = -1; // Reset current process
          }
        } else {
          currentTime++; // Increment time if no process is ready to execute
        }
    }

    // Print results
    printf("\nProcess\tArrival Time\tBurst Time\tPriority\tCompletion Time\tTurnaround Time\tWaiting Time\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].at, processes[i].orig_bt, processes[i].pt, processes[i].ct, processes[i].tat, processes[i].wt);
    }

    // Display Gantt chart
    printf("\nGantt Chart:\n");
    for (int i = 0; i < gantt_index; i++) {
        printf("  P%d  |", gantt[i]);
    }
    printf("\n");

    // Print time line
    int total_time = 0;
    printf("%-5d", total_time);
    for (int i = 0; i < gantt_index; i++) {
        total_time += gantt_time[i]; // Accumulate total time for each process
        printf("    %-5d", total_time);
    }
    printf("\n");
```

```
    free(processes); // Free allocated memory
    return 0;
}
```

```
Enter the number of processes: 4
Enter arrival time, burst time, and priority for process 1: 0 5 10
Enter arrival time, burst time, and priority for process 2: 1 4 20
Enter arrival time, burst time, and priority for process 3: 2 2 30
Enter arrival time, burst time, and priority for process 4: 4 1 40

Process Arrival Time    Burst Time      Priority        Completion Time Turnaround Time Waiting Time
P1              0               5               10              5               5               0
P2              1               4               20              9               8               4
P3              2               2               30              11              9               7
P4              4               1               40              12              8               7

Gantt Chart:
  P1  |  P2  |  P3  |  P4  |
0        5        9        11       12
```

Priority nonpreemptive:

```
#include <stdio.h>
#define MAX 10
// Structure to store process details
struct Process {
    int pid;  // Process ID
    int at;   // Arrival time
    int bt;   // Burst time
    int ct;   // Completion time
    int wt;   // Waiting time
    int tat;  // Turnaround time
    int pri;  // Priority
};

// Function to calculate waiting time and turnaround time
void calculateTimes(struct Process processes[], int n) {
    int completed = 0;
    int time = 0;
    int idx;
    int minPriority;
```

```
// Sorting based on arrival time
for (int i = 0; i < n - 1; i++) {
    for (int j = i + 1; j < n; j++) {
        if (processes[i].at > processes[j].at) {
            struct Process temp = processes[i];
            processes[i] = processes[j];
            processes[j] = temp;
        }
    }
}

// Gantt chart header
printf("\nGantt Chart:\n");
printf("Time -> ");

while (completed < n) {
    // Find the process with the highest priority (smallest priority number)
    minPriority = 9999;  // A large number
    idx = -1;
    for (int i = 0; i < n; i++) {
        if (processes[i].at <= time && processes[i].ct == 0) {
            if (processes[i].pri < minPriority) {
                minPriority = processes[i].pri;
                idx = i;
            }
        }
    }

    if (idx == -1) {
        time++;  // If no process is ready to execute, increment time
        continue;
```

```
        }

        // Process is executing
        printf("P%d ", processes[idx].pid);  // Print the process in the Gantt chart
        processes[idx].ct = time + processes[idx].bt;
        processes[idx].tat = processes[idx].ct - processes[idx].at;
        processes[idx].wt = processes[idx].tat - processes[idx].bt;

        // Increment time and mark the process as completed
        time += processes[idx].bt;
        completed++;
    }

    printf("\n\n");
}

// Function to display the results
void displayResults(struct Process processes[], int n) {
    printf("\nProcess ID\tArrival Time\tBurst Time\tPriority\tCompletion Time\tWaiting Time\tTurnaround Time\n");
    float totalWt = 0, totalTat = 0;

    for (int i = 0; i < n; i++) {
        totalWt += processes[i].wt;
        totalTat += processes[i].tat;
        printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
            processes[i].pid, processes[i].at, processes[i].bt, processes[i].pri, processes[i].ct, processes[i].wt, processes[i].tat);
    }

    printf("\nAverage Waiting Time: %.2f\n", totalWt / n);
    printf("Average Turnaround Time: %.2f\n", totalTat / n);
```

```
}

int main() {
    int n;

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

    struct Process processes[n];

    // Input process details
    for (int i = 0; i < n; i++) {
        processes[i].pid = i + 1;  // Process ID
        printf("Enter arrival time, burst time, and priority for Process %d: ", i + 1);
        scanf("%d %d %d", &processes[i].at, &processes[i].bt, &processes[i].pri);
    }

    // Calculate completion time, waiting time, and turnaround time
    calculateTimes(processes, n);

    // Display the results
    displayResults(processes, n);

    return 0;
}
```

```
Enter the number of processes: 4
Enter arrival time, burst time, and priority for Process 1: 0 4 2
Enter arrival time, burst time, and priority for Process 2: 1 3 1
Enter arrival time, burst time, and priority for Process 3: 2 6 3
Enter arrival time, burst time, and priority for Process 4: 3 2 4

Gantt Chart:
Time -> P1 P2 P3 P4


Process ID      Arrival Time    Burst Time      Priority        Completion Time Waiting Time    Turnaround Time
P1              0               4               2               4               0               4
P2              1               3               1               7               3               6
P3              2               6               3               13              5               11
P4              3               2               4               15              10              12
```

SRTF:

```
#include <stdio.h>

#define MAX 10

// Structure to store process details
struct Process {
   int pid;  // Process ID
   int at;   // Arrival time
   int bt;   // Burst time
   int ct;   // Completion time
   int wt;   // Waiting time
   int tat;  // Turnaround time
   int rt;   // Remaining time
};

// Function to find the process with the shortest remaining time
int findShortestRemainingTime(struct Process processes[], int n, int time) {
   int min = -1;
   int minIndex = -1;
   for (int i = 0; i < n; i++) {
      if (processes[i].at <= time && processes[i].rt > 0) {
         if (min == -1 || processes[i].rt < min) {
            min = processes[i].rt;
            minIndex = i;
         }
      }
   }
   return minIndex;
}
```

```
// Function to calculate waiting time and turnaround time
void calculateTimes(struct Process processes[], int n) {
    int time = 0;
    int completed = 0;

    // Initialize remaining time
    for (int i = 0; i < n; i++) {
        processes[i].rt = processes[i].bt;
    }

    // Loop to execute all processes
    while (completed < n) {
        // Find the process with the shortest remaining time
        int idx = findShortestRemainingTime(processes, n, time);
        if (idx == -1) {
            time++;  // If no process is ready to execute, just increment time
            continue;
        }

        // Process is executing
        processes[idx].rt--;  // Decrease the remaining time

        // If process is completed
        if (processes[idx].rt == 0) {
            completed++;
            processes[idx].ct = time + 1;
            processes[idx].tat = processes[idx].ct - processes[idx].at;
            processes[idx].wt = processes[idx].tat - processes[idx].bt;
        }
```

```
        time++;  // Move time forward
    }
}

// Function to display the results
void displayResults(struct Process processes[], int n) {
    printf("Process ID\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurnaround Time\n");
    float totalWt = 0, totalTat = 0;

    for (int i = 0; i < n; i++) {
        totalWt += processes[i].wt;
        totalTat += processes[i].tat;
        printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
               processes[i].pid, processes[i].at, processes[i].bt, processes[i].ct, processes[i].wt, processes[i].tat);
    }

    printf("\nAverage Waiting Time: %.2f\n", totalWt / n);
    printf("Average Turnaround Time: %.2f\n", totalTat / n);
}

int main() {
    int n;

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

    struct Process processes[n];

    // Input process details
```

```
    for (int i = 0; i < n; i++) {
        processes[i].pid = i + 1;  // Process ID
        printf("Enter arrival time and burst time for Process %d: ", i + 1);
        scanf("%d %d", &processes[i].at, &processes[i].bt);
    }

    // Calculate completion time, waiting time, and turnaround time
    calculateTimes(processes, n);

    // Display the results
    displayResults(processes, n);

    return 0;
}
```

```
Enter the number of processes: 4
Enter arrival time and burst time for Process 1: 0 8
Enter arrival time and burst time for Process 2: 1 4
Enter arrival time and burst time for Process 3: 2 9
Enter arrival time and burst time for Process 4: 3 5
Process ID      Arrival Time    Burst Time      Completion Time Waiting Time    Turnaround Time
P1              0               8               17              9               17
P2              1               4               5               0               4
P3              2               9               26              15              24
P4              3               5               10              2               7

Average Waiting Time: 6.50
Average Turnaround Time: 13.00
```

Process Synchronization:

Reader Writer problem:

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

int *buffer;
```

```
int count = 0;
int buffer_size;
sem_t empty;          // Semaphore for empty slots in the buffer
sem_t full;           // Semaphore for full slots in the buffer
pthread_mutex_t mutex; // Mutex to protect the critical section (buffer)

void *producer(void *arg) {
   int num_produce;
   printf("Enter number of items to produce: ");
   scanf("%d", &num_produce);

   for (int i = 0; i < num_produce; i++) {
      sem_wait(&empty);         // Wait for an empty slot in the buffer
      pthread_mutex_lock(&mutex);  // Lock the buffer (critical section)

      int item = rand() % 100;   // Produce an item (random number)
      buffer[count++] = item;     // Add item to buffer
      printf("Produced: %d | Buffer: ", item);

      // Display the current buffer state
      for (int j = 0; j < count; j++) {
         printf("%d ", buffer[j]);
      }
      printf("\n");

      pthread_mutex_unlock(&mutex);  // Unlock the buffer
      sem_post(&full);              // Signal that a new item is available
      sleep(1);                   // Sleep for 1 second
   }

   return NULL;
```

```
}

void *consumer(void *arg) {
    while (1) {
        char choice;
        printf("Do you want to consume an item? (y/n): ");
        scanf(" %c", &choice);  // Space before %c to consume the newline

        if (choice == 'y') {
            sem_wait(&full);          // Wait for a full slot in the buffer
            pthread_mutex_lock(&mutex); // Lock the buffer (critical section)

            int item = buffer[--count];  // Consume an item (decrement count)
            printf("Consumed: %d | Buffer: ", item);

            // Display the current buffer state
            for (int j = 0; j < count; j++) {
                printf("%d ", buffer[j]);
            }
            printf("\n");

            pthread_mutex_unlock(&mutex); // Unlock the buffer
            sem_post(&empty);           // Signal that a slot has become empty
        } else {
            break;  // Exit the loop if the user does not want to consume more
        }
    }
    return NULL;
}

int main() {
```

```
pthread_t prod, cons;

printf("Enter buffer size: ");
scanf("%d", &buffer_size);

// Dynamically allocate memory for the buffer based on user input
buffer = (int *)malloc(buffer_size * sizeof(int));
if (buffer == NULL) {
   perror("Failed to allocate buffer");
   return 1;
}

// Initialize semaphores and mutex
sem_init(&empty, 0, buffer_size); // Initially all slots are empty
sem_init(&full, 0, 0);            // Initially no items in the buffer
pthread_mutex_init(&mutex, NULL);  // Initialize the mutex

// Create the producer and consumer threads
pthread_create(&prod, NULL, producer, NULL);
pthread_create(&cons, NULL, consumer, NULL);

// Wait for the threads to finish
pthread_join(prod, NULL);
pthread_join(cons, NULL);

// Clean up semaphores and mutex
sem_destroy(&empty);
sem_destroy(&full);
pthread_mutex_destroy(&mutex);

// Free the dynamically allocated buffer
```

```
    free(buffer);
    return 0;
}
```

Producer Cosumer:

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

int *buffer;
int count = 0;
int buffer_size;
sem_t empty;          // Semaphore for empty slots in the buffer
sem_t full;           // Semaphore for full slots in the buffer
pthread_mutex_t mutex; // Mutex to protect the critical section (buffer)

void *producer(void *arg) {
    int num_produce;
    printf("Enter number of items to produce: ");
    scanf("%d", &num_produce);

    for (int i = 0; i < num_produce; i++) {
        sem_wait(&empty);         // Wait for an empty slot in the buffer
        pthread_mutex_lock(&mutex);  // Lock the buffer (critical section)

        int item = rand() % 100;   // Produce an item (random number)
        buffer[count++] = item;     // Add item to buffer
        printf("Produced: %d | Buffer: ", item);
```

```
        // Display the current buffer state
        for (int j = 0; j < count; j++) {
            printf("%d ", buffer[j]);
        }
        printf("\n");

        pthread_mutex_unlock(&mutex);  // Unlock the buffer
        sem_post(&full);               // Signal that a new item is available
        sleep(1);                      // Sleep for 1 second
    }

    return NULL;
}

void *consumer(void *arg) {
    while (1) {
        char choice;
        printf("Do you want to consume an item? (y/n): ");
        scanf(" %c", &choice);  // Space before %c to consume the newline

        if (choice == 'y') {
            sem_wait(&full);          // Wait for a full slot in the buffer
            pthread_mutex_lock(&mutex); // Lock the buffer (critical section)

            int item = buffer[--count];  // Consume an item (decrement count)
            printf("Consumed: %d | Buffer: ", item);

            // Display the current buffer state
            for (int j = 0; j < count; j++) {
                printf("%d ", buffer[j]);
```

```
            }
            printf("\n");

            pthread_mutex_unlock(&mutex); // Unlock the buffer
            sem_post(&empty);           // Signal that a slot has become empty
        } else {
            break;  // Exit the loop if the user does not want to consume more
        }
    }
    return NULL;
}

int main() {
    pthread_t prod, cons;

    printf("Enter buffer size: ");
    scanf("%d", &buffer_size);

    // Dynamically allocate memory for the buffer based on user input
    buffer = (int *)malloc(buffer_size * sizeof(int));
    if (buffer == NULL) {
        perror("Failed to allocate buffer");
        return 1;
    }

    // Initialize semaphores and mutex
    sem_init(&empty, 0, buffer_size); // Initially all slots are empty
    sem_init(&full, 0, 0);           // Initially no items in the buffer
    pthread_mutex_init(&mutex, NULL);  // Initialize the mutex

    // Create the producer and consumer threads
```

```
    pthread_create(&prod, NULL, producer, NULL);
    pthread_create(&cons, NULL, consumer, NULL);

    // Wait for the threads to finish
    pthread_join(prod, NULL);
    pthread_join(cons, NULL);

    // Clean up semaphores and mutex
    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&mutex);

    // Free the dynamically allocated buffer
    free(buffer);
    return 0;
}
```

Dinning Philosopher:

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

#define NUM_PHILOSOPHERS 5

pthread_mutex_t forks[NUM_PHILOSOPHERS];  // Mutexes for each fork

void* philosopher(void* num) {
    int phil = *((int*) num); // Philosopher number

    while(1) {
```

```
        printf("Philosopher %d is thinking.\n", phil);
        sleep(rand() % 3); // Thinking for a random time

        // Pick up left fork
        pthread_mutex_lock(&forks[phil]);
        printf("Philosopher %d picked up left fork %d.\n", phil, phil);

        // Pick up right fork
        pthread_mutex_lock(&forks[(phil + 1) % NUM_PHILOSOPHERS]);
        printf("Philosopher %d picked up right fork %d.\n", phil, (phil + 1) % NUM_PHILOSOPHERS);

        // Eat
        printf("Philosopher %d is eating.\n", phil);
        sleep(rand() % 3); // Eating for a random time

        // Put down right fork
        pthread_mutex_unlock(&forks[(phil + 1) % NUM_PHILOSOPHERS]);
        printf("Philosopher %d put down right fork %d.\n", phil, (phil + 1) % NUM_PHILOSOPHERS);

        // Put down left fork
        pthread_mutex_unlock(&forks[phil]);
        printf("Philosopher %d put down left fork %d.\n", phil, phil);
    }

    return NULL;
}

int main() {
    pthread_t philosophers[NUM_PHILOSOPHERS];
    int philosopher_nums[NUM_PHILOSOPHERS];
```

```
    // Initialize the mutexes for each fork
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_mutex_init(&forks[i], NULL);
    }

    // Create philosopher threads
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        philosopher_nums[i] = i;
        pthread_create(&philosophers[i], NULL, philosopher, (void*)&philosopher_nums[i]);
    }




    // Join philosopher threads
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_join(philosophers[i], NULL);
    }

    // Destroy the mutexes
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_mutex_destroy(&forks[i]);
    }

    return 0;
}
```

Deadlock:

Bankers Algo:

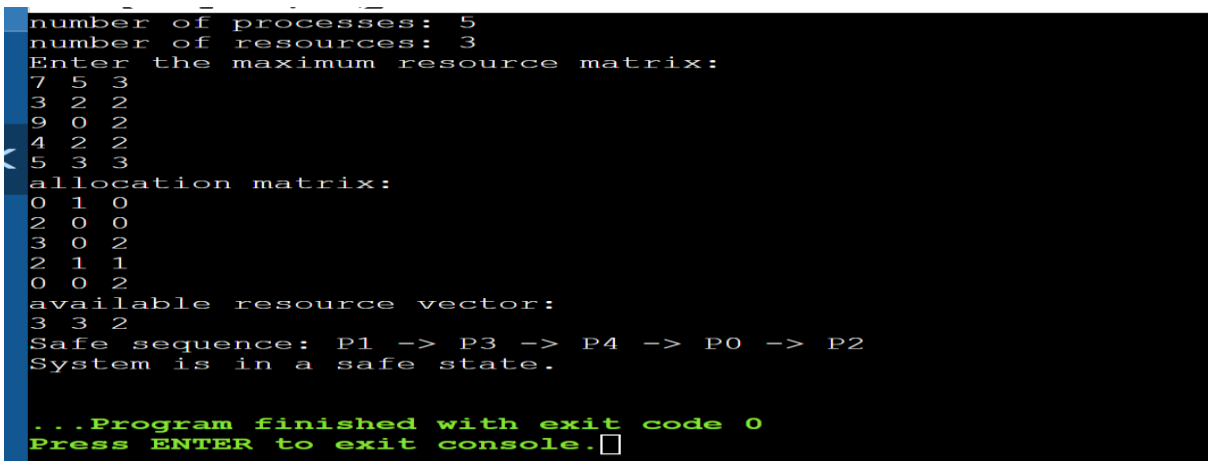
```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
```

```
bool isSafe(int** max, int** alloc, int* avail, int process, int numResources) {
    int* need = (int*)malloc(numResources * sizeof(int));
    for (int i = 0; i < numResources; i++) {
        need[i] = max[process][i] - alloc[process][i];
    }

    for (int i = 0; i < numResources; i++) {
        if (need[i] > avail[i]) {
            free(need); // Free the allocated memory for 'need'
            return false;
        }
    }

    free(need); // Free the allocated memory for 'need'
    return true;
}

bool bankersAlgorithm(int** max, int** alloc, int* avail, int numProcesses, int numResources) {
    bool* finish = (bool*)malloc(numProcesses * sizeof(bool));
    int* safeSequence = (int*)malloc(numProcesses * sizeof(int));
    int numFinished = 0;

    for (int i = 0; i < numProcesses; i++) {
        finish[i] = false;
    }

    int safeSequenceIndex = 0;
    while (numFinished < numProcesses) {
        bool found = false;
        for (int i = 0; i < numProcesses; i++) {
```

```
            if (!finish[i] && isSafe(max, alloc, avail, i, numResources)) {
                for (int j = 0; j < numResources; j++) {
                    avail[j] += alloc[i][j];
                }
                safeSequence[safeSequenceIndex++] = i;
                finish[i] = true;
                numFinished++;
                found = true;
            }
        }
        if (!found) {
            free(finish);
            free(safeSequence);
            return false;
        }
    }

    printf("Safe sequence: ");
    for (int i = 0; i < safeSequenceIndex; i++) {
        printf("P%d", safeSequence[i]);
        if (i < safeSequenceIndex - 1) {
            printf(" -> ");
        }
    }
    printf("\n");

    free(finish);
    free(safeSequence);
    return true;
}
```

```
int main() {
    int numProcesses, numResources;

    printf("Number of processes: ");
    scanf("%d", &numProcesses);
    printf("Number of resources: ");
    scanf("%d", &numResources);

    int** max = (int**)malloc(numProcesses * sizeof(int*));
    int** alloc = (int**)malloc(numProcesses * sizeof(int*));
    int* avail = (int*)malloc(numResources * sizeof(int));

    for (int i = 0; i < numProcesses; i++) {
        max[i] = (int*)malloc(numResources * sizeof(int));
        alloc[i] = (int*)malloc(numResources * sizeof(int));
    }

    printf("Enter the maximum resource matrix:\n");
    for (int i = 0; i < numProcesses; i++) {
        for (int j = 0; j < numResources; j++) {
            scanf("%d", &max[i][j]);
        }
    }

    printf("Enter the allocation matrix:\n");
    for (int i = 0; i < numProcesses; i++) {
        for (int j = 0; j < numResources; j++) {
            scanf("%d", &alloc[i][j]);
        }
    }
```

```
    printf("Enter the available resource vector:\n");
    for (int i = 0; i < numResources; i++) {
        scanf("%d", &avail[i]);
    }

    if (bankersAlgorithm(max, alloc, avail, numProcesses, numResources)) {
        printf("System is in a safe state.\n");
    } else {
        printf("System is in an unsafe state.\n");
    }

    for (int i = 0; i < numProcesses; i++) {
        free(max[i]);
        free(alloc[i]);
    }

    free(max);
    free(alloc);
    free(avail);

    return 0;
}
```



Deadlock detection:

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

bool allFinished(bool* finished, int n) {
   for (int i = 0; i < n; i++) {
      if (!finished[i]) {
         return false;
      }
   }
   return true;
}

bool check(int* request, int* avail, int m) {
   for (int i = 0; i < m; i++) {
      if (request[i] > avail[i]) {
         return false;
      }
   }
   return true;
}

int main() {
   int n, m;
   printf("Enter the number of processes: ");
   scanf("%d", &n);
   printf("Enter the number of resource types: ");
   scanf("%d", &m);
```

```
int work[m];
printf("Enter the available resources of each type:\n");
for (int i = 0; i < m; i++) {
   scanf("%d", &work[i]);
}

int allocation[n][m];
int request[n][m];

// Reading the allocation matrix
for (int i = 0; i < n; i++) {
   printf("Enter allocation for process %d\n", i + 1);
   for (int j = 0; j < m; j++) {
      scanf("%d", &allocation[i][j]);
   }
}

// Reading the request matrix
for (int i = 0; i < n; i++) {
   printf("Enter request for process %d\n", i + 1);
   for (int j = 0; j < m; j++) {
      scanf("%d", &request[i][j]);
   }
}

bool finished[n];
for (int i = 0; i < n; i++) {
   finished[i] = false;
}
```

```
int seq[n];
int seqIndex = 0;

while (1) {
   bool found = false;

   for (int j = 0; j < n; j++) {
      if (!finished[j] && check(request[j], work, m)) {
         found = true;
         for (int k = 0; k < m; k++) {
            work[k] += allocation[j][k];
         }
         finished[j] = true;
         seq[seqIndex++] = j;
      }
   }

   if (!found) {
      printf("Deadlock detected\n");
      return 0;
   }

   if (allFinished(finished, n)) {
      break;
   }
}

printf("Deadlock is not present\n");
printf("Safe sequence: ");
for (int i = 0; i < seqIndex; i++) {
   printf("%d", seq[i] + 1);  // 1-based indexing for process number
```

```
        if (i < seqIndex - 1) {
            printf(" -> ");
        }
    }
    printf("\n");

    return 0;
}
```

```
Enter the number of processes: 3
Enter the number of resource types: 3
Enter the available resources of each type:
3 3 2
Enter allocation for process 1
0 1 0
Enter allocation for process 2
2 0 0
Enter allocation for process 3
3 1 2
Enter request for process 1
1 0 2
Enter request for process 2
2 0 0
Enter request for process 3
0 1 1
Deadlock is not present
Safe sequence: 1 -> 2 -> 3
```

Disk Scheduling:

FCFS

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

int main() {
    int n, st, dist = 0;
    printf("Enter the number of disk requests: ");
    scanf("%d", &n);

    int requests[n];
    printf("Enter the disk requests: ");
    for(int i = 0; i < n; i++) {
```

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

    printf("Enter the initial head position: ");
    scanf("%d", &st);

    int curr = st;
    printf("\nDisk Scheduling Order (FCFS):\n");
    for(int i = 0; i < n; i++) {
        printf("Move from %d to %d, distance = %d\n", curr, requests[i], abs(requests[i] - curr));
        dist += abs(requests[i] - curr);
        curr = requests[i];
    }

    printf("\nTotal distance traveled by the disk arm: %d\n", dist);
    printf("Average Seek Distance: %.2f\n", (float)dist / n);

    return 0;
}
```

```
Enter the number of disk requests: 7
Enter the disk requests: 82 170 43
140 24 16 190
Enter the initial head position: 50

Disk Scheduling Order (FCFS):
Move from 50 to 82, distance = 32
Move from 82 to 170, distance = 88
Move from 170 to 43, distance = 127
Move from 43 to 140, distance = 97
Move from 140 to 24, distance = 116
Move from 24 to 16, distance = 8
Move from 16 to 190, distance = 174

Total distance traveled by the disk arm: 642
Average Seek Distance: 91.71
```

SSTF:

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

```
int findClosestRequest(int curr, int requests[], int n, int processed[]) {
    int minDistance = 1000000; // Arbitrary large number for comparison
    int index = -1;

    for (int i = 0; i < n; i++) {
        if (processed[i] == 0) {  // Unprocessed request
            int distance = abs(curr - requests[i]);
            if (distance < minDistance) {
                minDistance = distance;
                index = i;
            }
        }
    }
    return index;
}

int main() {
    int n, st, dist = 0;

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

    int requests[n];
    printf("Enter the disk requests: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }

    printf("Enter the initial head position: ");
    scanf("%d", &st);
```

```
    int curr = st;
    int completed = 0;
    int processed[n];

    for (int i = 0; i < n; i++) {
        processed[i] = 0;  // Initialize all requests as unprocessed
    }

    printf("\nDisk Scheduling Order (SSTF):\n");

    while (completed < n) {
        int index = findClosestRequest(curr, requests, n, processed);

        // If all requests have been processed, break out
        if (index == -1) {
            break;
        }

        // Process the closest request
        printf("Move from %d to %d, distance = %d\n", curr, requests[index], abs(curr -
requests[index]));
        dist += abs(curr - requests[index]);
        curr = requests[index];
        processed[index] = 1;  // Mark the request as processed
        completed++;
    }
    printf("\nTotal distance traveled by the disk arm: %d\n", dist);
    printf("Average Seek Distance: %.2f\n", (float)dist / n);
return 0;
```

}

```
Enter the number of disk requests: 7
Enter the disk requests: 82 170 43 140 24 16 190
Enter the initial head position: 50

Disk Scheduling Order (SSTF):
Move from 50 to 43, distance = 7
Move from 43 to 24, distance = 19
Move from 24 to 16, distance = 8
Move from 16 to 82, distance = 66
Move from 82 to 140, distance = 58
Move from 140 to 170, distance = 30
Move from 170 to 190, distance = 20

Total distance traveled by the disk arm: 208
Average Seek Distance: 29.71
```

SCAN:

```
#include <stdio.h>

#include <stdlib.h>


void scan(int requests[], int n, int st, int disk_size) {
    int left = 0, right = 0;
    int distance = 0;
    int total_distance = 0;


    int left_arr[n], right_arr[n];
    int left_index = 0, right_index = 0;


    // Separate the requests into left and right of the initial head position
    for (int i = 0; i < n; i++) {
        if (requests[i] < st) {
            left_arr[left_index++] = requests[i];
        } else {
            right_arr[right_index++] = requests[i];
        }
    }


    // Sort the left and right arrays
```

```
for (int i = 0; i < left_index - 1; i++) {
    for (int j = i + 1; j < left_index; j++) {
        if (left_arr[i] < left_arr[j]) {
            int temp = left_arr[i];
            left_arr[i] = left_arr[j];
            left_arr[j] = temp;
        }
    }
}

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

// Total distance calculation
// Traverse the right side of the head
total_distance += abs(st - right_arr[0]);
distance = total_distance;
for (int i = 0; i < right_index; i++) {
    printf("%d -> ", right_arr[i]);
    if (i == right_index - 1) {
        total_distance += abs(right_arr[i] - disk_size);
        distance = total_distance;
    } else {
        total_distance += abs(right_arr[i] - right_arr[i + 1]);
```

```
        }
    }

    // Now reverse and traverse the left side of the head
    total_distance += abs(disk_size - left_arr[0]);
    for (int i = 0; i < left_index; i++) {
        printf("%d -> ", left_arr[i]);
        if (i == left_index - 1) {
            total_distance += abs(left_arr[i] - 0);
            distance = total_distance;
        } else {
            total_distance += abs(left_arr[i] - left_arr[i + 1]);
        }
    }

    printf("\nTotal distance: %d\n", total_distance);
    printf("Average Seek Distance: %.2f\n", (float)total_distance / n);
}

int main() {
    int n, st, disk_size;

    // Input the number of requests, initial head position, and disk size
    printf("Enter the number of disk requests: ");
    scanf("%d", &n);

    int requests[n];
    printf("Enter the disk requests: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }
```

```
    printf("Enter the initial head position: ");
    scanf("%d", &st);

    printf("Enter the disk size: ");
    scanf("%d", &disk_size);

    // Call SCAN function
    scan(requests, n, st, disk_size);

    return 0;
}
```

```
Enter the number of disk requests: 7
Enter the disk requests: 82 170 43 140 24 16 190
Enter the initial head position: 50
Enter the disk size: 200
82 -> 140 -> 170 -> 190 -> 43 -> 24 -> 16 ->
Total distance: 350
Average Seek Distance: 50.00
```

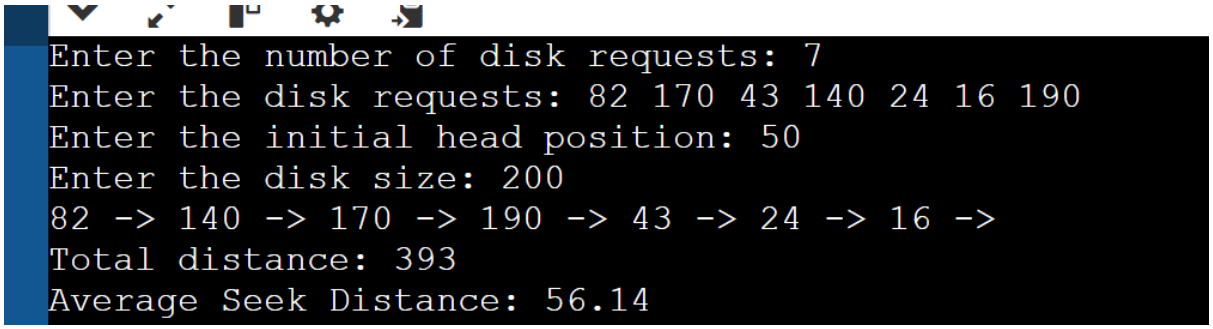
CSCAN:

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

void cscan(int requests[], int n, int st, int disk_size) {
    int left = 0, right = 0;
    int distance = 0;
    int total_distance = 0;

    int left_arr[n], right_arr[n];
    int left_index = 0, right_index = 0;
```

```
// Separate the requests into left and right of the initial head position
for (int i = 0; i < n; i++) {
    if (requests[i] < st) {
        left_arr[left_index++] = requests[i];
    } else {
        right_arr[right_index++] = requests[i];
    }
}

// Sort the left and right arrays
for (int i = 0; i < left_index - 1; i++) {
    for (int j = i + 1; j < left_index; j++) {
        if (left_arr[i] < left_arr[j]) {
            int temp = left_arr[i];
            left_arr[i] = left_arr[j];
            left_arr[j] = temp;
        }
    }
}

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

// Total distance calculation
```

```
    // Traverse the right side of the head
    total_distance += abs(st - right_arr[0]);
    distance = total_distance;
    for (int i = 0; i < right_index; i++) {
        printf("%d -> ", right_arr[i]);
        if (i == right_index - 1) {
            total_distance += abs(right_arr[i] - disk_size);
            distance = total_distance;
        } else {
            total_distance += abs(right_arr[i] - right_arr[i + 1]);
        }
    }

    // After reaching the maximum, go to the start of the disk (0) and process the left side
    total_distance += abs(disk_size - 0);
    for (int i = 0; i < left_index; i++) {
        printf("%d -> ", left_arr[i]);
        if (i == left_index - 1) {
            total_distance += abs(left_arr[i] - 0);
            distance = total_distance;
        } else {
            total_distance += abs(left_arr[i] - left_arr[i + 1]);
        }
    }

    printf("\nTotal distance: %d\n", total_distance);
    printf("Average Seek Distance: %.2f\n", (float)total_distance / n);
}

int main() {
    int n, st, disk_size;
```

```
    // Input the number of requests, initial head position, and disk size
    printf("Enter the number of disk requests: ");
    scanf("%d", &n);

    int requests[n];
    printf("Enter the disk requests: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }

    printf("Enter the initial head position: ");
    scanf("%d", &st);

    printf("Enter the disk size: ");
    scanf("%d", &disk_size);

    // Call C-SCAN function
    cscan(requests, n, st, disk_size);

    return 0;
}
```



LOOK:

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

```
#include <stdlib.h>

void look(int requests[], int n, int st, int disk_size) {
    int left = 0, right = 0;
    int distance = 0;
    int total_distance = 0;

    int left_arr[n], right_arr[n];
    int left_index = 0, right_index = 0;

    // Separate the requests into left and right of the initial head position
    for (int i = 0; i < n; i++) {
        if (requests[i] < st) {
            left_arr[left_index++] = requests[i];
        } else {
            right_arr[right_index++] = requests[i];
        }
    }

    // Sort the left and right arrays
    for (int i = 0; i < left_index - 1; i++) {
        for (int j = i + 1; j < left_index; j++) {
            if (left_arr[i] < left_arr[j]) {
                int temp = left_arr[i];
                left_arr[i] = left_arr[j];
                left_arr[j] = temp;
            }
        }
    }

    for (int i = 0; i < right_index - 1; i++) {
```

```
    for (int j = i + 1; j < right_index; j++) {
        if (right_arr[i] > right_arr[j]) {
            int temp = right_arr[i];
            right_arr[i] = right_arr[j];
            right_arr[j] = temp;
        }
    }
}

// Traverse the right side of the head
total_distance += abs(st - right_arr[0]);
distance = total_distance;
for (int i = 0; i < right_index; i++) {
    printf("%d -> ", right_arr[i]);
    if (i == right_index - 1) {
        total_distance += abs(right_arr[i] - left_arr[0]); // Change direction after the last right request
        distance = total_distance;
    } else {
        total_distance += abs(right_arr[i] - right_arr[i + 1]);
    }
}

// Traverse the left side of the head
for (int i = 0; i < left_index; i++) {
    printf("%d -> ", left_arr[i]);
    if (i == left_index - 1) {
        total_distance += abs(left_arr[i] - left_arr[0]);
        distance = total_distance;
    } else {
        total_distance += abs(left_arr[i] - left_arr[i + 1]);
    }
```

```
    }

    printf("\nTotal distance: %d\n", total_distance);
    printf("Average Seek Distance: %.2f\n", (float)total_distance / n);
}

int main() {
    int n, st, disk_size;

    // Input the number of requests, initial head position, and disk size
    printf("Enter the number of disk requests: ");
    scanf("%d", &n);

    int requests[n];
    printf("Enter the disk requests: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }

    printf("Enter the initial head position: ");
    scanf("%d", &st);

    printf("Enter the disk size: ");
    scanf("%d", &disk_size);

    // Call LOOK function
    look(requests, n, st, disk_size);

    return 0;
}
```

```
Enter the number of disk requests: 7
Enter the disk requests: 82 170 43 140 24 16 190
Enter the initial head position: 50
Enter the disk size: 200
```

C-LOOK:

```
#include <stdio.h>

#include <stdlib.h>


void clook(int requests[], int n, int st, int disk_size) {
    int left = 0, right = 0;
    int distance = 0;
    int total_distance = 0;

    int left_arr[n], right_arr[n];
    int left_index = 0, right_index = 0;

    // Separate the requests into left and right of the initial head position
    for (int i = 0; i < n; i++) {
        if (requests[i] < st) {
            left_arr[left_index++] = requests[i];
        } else {
            right_arr[right_index++] = requests[i];
        }
    }

    // Sort the left and right arrays
    for (int i = 0; i < left_index - 1; i++) {
        for (int j = i + 1; j < left_index; j++) {
            if (left_arr[i] < left_arr[j]) {
                int temp = left_arr[i];
                left_arr[i] = left_arr[j];
```

```
            left_arr[j] = temp;
        }
    }
}

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

// Traverse the right side of the head
total_distance += abs(st - right_arr[0]);
distance = total_distance;
for (int i = 0; i < right_index; i++) {
    printf("%d -> ", right_arr[i]);
    if (i == right_index - 1) {
        total_distance += abs(right_arr[i] - left_arr[0]); // Change direction after the last right request
        distance = total_distance;
    } else {
        total_distance += abs(right_arr[i] - right_arr[i + 1]);
    }
}

// Traverse the left side of the head
for (int i = 0; i < left_index; i++) {
    printf("%d -> ", left_arr[i]);
```

```
        if (i == left_index - 1) {
            total_distance += abs(left_arr[i] - left_arr[0]);
            distance = total_distance;
        } else {
            total_distance += abs(left_arr[i] - left_arr[i + 1]);
        }
    }
  printf("\nTotal distance: %d\n", total_distance);
    printf("Average Seek Distance: %.2f\n", (float)total_distance / n);
}

int main() {
    int n, st, disk_size;

    // Input the number of requests, initial head position, and disk size
    printf("Enter the number of disk requests: ");
    scanf("%d", &n);

    int requests[n];
    printf("Enter the disk requests: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }

    printf("Enter the initial head position: ");
    scanf("%d", &st);

    printf("Enter the disk size: ");
    scanf("%d", &disk_size);

    // Call C-LOOK function
```

```
    clook(requests, n, st, disk_size);

    return 0;
}
```

```
Enter the number of disk requests: 7
Enter the disk requests: 82 170 43 140 24 16 190
Enter the initial head position: 50
Enter the disk size: 200
82 -> 140 -> 170 -> 190 -> 43 -> 24 -> 16 ->
Total distance: 341
Average Seek Distance: 48.71
```

Memory management:

First Fit:

```
#include <stdio.h>

void firstFit(int blockSizes[], int m, int processSizes[], int n) {
    int allocation[n];

    // Initialize allocation as -1 (not allocated)
    for (int i = 0; i < n; i++) {
        allocation[i] = -1;
    }

    // Iterate over each process
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            // Find the first block that can fit the process
            if (blockSizes[j] >= processSizes[i]) {
                allocation[i] = j;
                blockSizes[j] -= processSizes[i]; // Reduce available block size
                break;
            }
```

```
        }
    }

    // Display allocation
    printf("\nProcess No.\tProcess Size\tBlock Allocated\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t", i + 1, processSizes[i]);
        if (allocation[i] != -1) {
            printf("%d\n", allocation[i] + 1);
        } else {
            printf("Not Allocated\n");
        }
    }
}

int main() {
    int m, n;

    // User input for number of blocks and processes
    printf("Enter the number of memory blocks: ");
    scanf("%d", &m);
    int blockSizes[m];
    printf("Enter the sizes of the memory blocks:\n");
    for (int i = 0; i < m; i++) {
        scanf("%d", &blockSizes[i]);
    }

    printf("Enter the number of processes: ");
    scanf("%d", &n);
    int processSizes[n];
    printf("Enter the sizes of the processes:\n");
```

```
    for (int i = 0; i < n; i++) {
        scanf("%d", &processSizes[i]);
    }

    // Call First Fit function
    firstFit(blockSizes, m, processSizes, n);

    return 0;
}
```

```
Enter the number of memory blocks: 5
Enter the sizes of the memory blocks:
100 500 200 300 600
Enter the number of processes: 4
Enter the sizes of the processes:
212 417 112 426

Process No.     Process Size    Block Allocated
1               212             2
2               417             5
3               112             2
4               426             Not Allocated
```

Next Fit:

```
#include <stdio.h>

void nextFit(int blockSizes[], int m, int processSizes[], int n) {
    int allocation[n]; // To track block allocated to each process
    int lastAllocated = 0; // To track the last block index used for allocation

    // Initialize all allocations to -1 (not allocated)
    for (int i = 0; i < n; i++) {
        allocation[i] = -1;
    }
```

```
    // Iterate over each process to allocate memory
    for (int i = 0; i < n; i++) {
        int startIndex = lastAllocated; // Start from the last allocated block
        do {
            if (blockSizes[lastAllocated] >= processSizes[i]) {
                allocation[i] = lastAllocated; // Allocate the block to this process
                blockSizes[lastAllocated] -= processSizes[i]; // Reduce the block size
                break; // Process allocated, move to the next process
            }
            lastAllocated = (lastAllocated + 1) % m; // Move to the next block (circular)
        } while (lastAllocated != startIndex); // Stop if we've looped back to the starting block
    }

    // Display the results
    printf("\nProcess No.\tProcess Size\tBlock Allocated\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t", i + 1, processSizes[i]);
        if (allocation[i] != -1) {
            printf("%d\n", allocation[i] + 1); // Display 1-based block index
        } else {
            printf("Not Allocated\n");
        }
    }
}

int main() {
    int m, n;

    // Input for number of memory blocks and their sizes
    printf("Enter the number of memory blocks: ");
    scanf("%d", &m);
```

```
    int blockSizes[m];
    printf("Enter the sizes of the memory blocks:\n");
    for (int i = 0; i < m; i++) {
        scanf("%d", &blockSizes[i]);
    }

    // Input for number of processes and their sizes
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    int processSizes[n];
    printf("Enter the sizes of the processes:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &processSizes[i]);
    }

    // Call Next Fit allocation
    nextFit(blockSizes, m, processSizes, n);

    return 0;
}
```

mathematica Copy code

```
Enter the number of memory blocks: 5
Enter the sizes of the memory blocks:
100 500 200 300 600
Enter the number of processes: 4
Enter the sizes of the processes:
212 417 112 426
```

Output:

arduino Copy code

```
Process No.    Process Size    Block Allocated
1              212             3
2              417             5
3              112             4
4              426             Not Allocated
```

Best Fit:

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

```
#include <limits.h> // For INT_MAX

void bestFit(int blockSizes[], int m, int processSizes[], int n) {
    int allocation[n]; // To track block allocated to each process

    // Initialize all allocations to -1 (not allocated)
    for (int i = 0; i < n; i++) {
        allocation[i] = -1;
    }

    // Iterate over each process to allocate memory
    for (int i = 0; i < n; i++) {
        int bestIdx = -1; // Initialize the best fit block index
        for (int j = 0; j < m; j++) {
            if (blockSizes[j] >= processSizes[i]) {
                if (bestIdx == -1 || blockSizes[j] < blockSizes[bestIdx]) {
                    bestIdx = j; // Update the best fit block index
                }
            }
        }

        // If a suitable block is found
        if (bestIdx != -1) {
            allocation[i] = bestIdx; // Allocate the block to the process
            blockSizes[bestIdx] -= processSizes[i]; // Reduce the block size
        }
    }

    // Display the results
    printf("\nProcess No.\tProcess Size\tBlock Allocated\n");
    for (int i = 0; i < n; i++) {
```

```
        printf("%d\t\t%d\t\t", i + 1, processSizes[i]);
        if (allocation[i] != -1) {
            printf("%d\n", allocation[i] + 1); // Display 1-based block index
        } else {
            printf("Not Allocated\n");
        }
    }
}

int main() {
    int m, n;
// Input for number of memory blocks and their sizes
    printf("Enter the number of memory blocks: ");
    scanf("%d", &m);
    int blockSizes[m];
    printf("Enter the sizes of the memory blocks:\n");
    for (int i = 0; i < m; i++) {
        scanf("%d", &blockSizes[i]);
    }

    // Input for number of processes and their sizes
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    int processSizes[n];
    printf("Enter the sizes of the processes:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &processSizes[i]);
    }

    // Call Best Fit allocation
    bestFit(blockSizes, m, processSizes, n);
```

```
    return 0;
}
```

```
Enter the number of memory blocks: 5
Enter the sizes of the memory blocks:
100 500 200 300 600
Enter the number of processes: 4
Enter the sizes of the processes:
212 417 112 426

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

Worst Fit:

```
#include <stdio.h>


void worstFit(int blockSizes[], int m, int processSizes[], int n) {
    int allocation[n]; // To track block allocated to each process

    // Initialize all allocations to -1 (not allocated)
    for (int i = 0; i < n; i++) {
        allocation[i] = -1;
    }

    // Iterate over each process to allocate memory
    for (int i = 0; i < n; i++) {
        int worstIdx = -1; // Initialize the worst fit block index
        for (int j = 0; j < m; j++) {
            if (blockSizes[j] >= processSizes[i]) {
                if (worstIdx == -1 || blockSizes[j] > blockSizes[worstIdx]) {
                    worstIdx = j; // Update the worst fit block index
                }
            }
        }
```

```
        // If a suitable block is found
        if (worstIdx != -1) {
            allocation[i] = worstIdx; // Allocate the block to the process
            blockSizes[worstIdx] -= processSizes[i]; // Reduce the block size
        }
    }

    // Display the results
    printf("\nProcess No.\tProcess Size\tBlock Allocated\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t", i + 1, processSizes[i]);
        if (allocation[i] != -1) {
            printf("%d\n", allocation[i] + 1); // Display 1-based block index
        } else {
            printf("Not Allocated\n");
        }
    }
}

int main() {
    int m, n;

    // Input for number of memory blocks and their sizes
    printf("Enter the number of memory blocks: ");
    scanf("%d", &m);
    int blockSizes[m];
    printf("Enter the sizes of the memory blocks:\n");
    for (int i = 0; i < m; i++) {
        scanf("%d", &blockSizes[i]);
    }
```

```
    // Input for number of processes and their sizes
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    int processSizes[n];
    printf("Enter the sizes of the processes:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &processSizes[i]);
    }

    // Call Worst Fit allocation
    worstFit(blockSizes, m, processSizes, n);

    return 0;
}
```

```
Enter the number of memory blocks: 5
Enter the sizes of the memory blocks:
100 500 200 300 600
Enter the number of processes: 4
Enter the sizes of the processes:
212 417 112 426

Process No.      Process Size     Block Allocated
1                212              5
2                417              2
3                112              5
4                426              Not Allocated
```

Shell Script:

Write a shell script program for different string handling functions.

```
#!/bin/bash
# Function to find the length of a string
string_length() {
    echo "Enter a string:"
    read str
    echo "Length of the string is: ${#str}"
}
```

```
# Function to compare two strings
string_compare() {
   echo "Enter first string:"
   read str1
   echo "Enter second string:"
   read str2
  if [ "$str1" == "$str2" ]; then
      echo "Strings are equal."
   else
      echo "Strings are not equal."
   fi
}
# Function to concatenate two strings
string_concatenate() {
   echo "Enter first string:"
   read str1
   echo "Enter second string:"
   read str2
   echo "Concatenated string: $str1$str2"
}

# Function to extract a substring
string_substring() {
   echo "Enter a string:"
   read str
   echo "Enter the starting position (0-based index):"
   read start
   echo "Enter the length of the substring:"
   read length
   echo "Extracted substring: ${str:$start:$length}"
}
```

```
# Function to reverse a string
string_reverse() {
   echo "Enter a string:"
   read str
   echo "Reversed string: $(echo $str | rev)"
}

# Function to check if a string contains a substring
string_contains() {
   echo "Enter the main string:"
   read main
   echo "Enter the substring to search for:"
   read sub
   if [[ $main == *"$sub"* ]]; then
      echo "Substring found."
   else
      echo "Substring not found."
   fi
}

# Main menu
while true; do
   echo ""
   echo "String Handling Functions"
   echo "1. Find string length"
   echo "2. Compare strings"
   echo "3. Concatenate strings"
   echo "4. Extract substring"
   echo "5. Reverse string"
   echo "6. Check if string contains substring"
```

```
    echo "7. Exit"
    echo "Choose an option:"
    read choice

    case $choice in
        1) string_length ;;
        2) string_compare ;;
        3) string_concatenate ;;
        4) string_substring ;;
        5) string_reverse ;;
        6) string_contains ;;
        7) echo "Exiting..."; exit 0 ;;
        *) echo "Invalid choice. Please try again." ;;
    esac
done
```

```
This message is shown once a day. To disable it please create the
/home/sumit/.hushlogin file.
sumit@LAPTOP-IU1P54PB:~$ nano median.sh
sumit@LAPTOP-IU1P54PB:~$ chmod+x median.sh
chmod+x: command not found
sumit@LAPTOP-IU1P54PB:~$ chmod +x median.sh
sumit@LAPTOP-IU1P54PB:~$ ./median.sh

String Handling Functions
1. Find string length
2. Compare strings
3. Concatenate strings
4. Extract substring
5. Reverse string
6. Check if string contains substring
7. Exit
Choose an option:
1
Enter a string:
Sumit
Length of the string is: 5
```

Write a shell script program for different arithmetic functions, example-grading systems

```
#!/bin/bash

# Function for basic arithmetic operations
arithmetic_operations() {
```

```
    echo "Enter the first number:"
    read num1
    echo "Enter the second number:"
    read num2

    echo "Choose an operation:"
    echo "1. Addition"
    echo "2. Subtraction"
    echo "3. Multiplication"
    echo "4. Division"
    echo "5. Modulus"
    read operation

    case $operation in
        1) echo "Result: $((num1 + num2))" ;;
        2) echo "Result: $((num1 - num2))" ;;
        3) echo "Result: $((num1 * num2))" ;;
        4)
            if [ $num2 -eq 0 ]; then
                echo "Error: Division by zero is not allowed."
            else
                echo "Result: $((num1 / num2))"
            fi
            ;;
        5) echo "Result: $((num1 % num2))" ;;
        *) echo "Invalid operation." ;;
    esac
}

# Function for grading system
grading_system() {
```

```
    echo "Enter the marks (out of 100):"
    read marks

    if [ $marks -ge 90 ]; then
        echo "Grade: A"
    elif [ $marks -ge 80 ]; then
        echo "Grade: B"
    elif [ $marks -ge 70 ]; then
        echo "Grade: C"
    elif [ $marks -ge 60 ]; then
        echo "Grade: D"
    elif [ $marks -ge 40 ]; then
        echo "Grade: E"
    else
        echo "Grade: F (Fail)"
    fi
}

# Main menu
while true; do
    echo ""
    echo "Arithmetic Functions and Grading System"
    echo "1. Perform Arithmetic Operations"
    echo "2. Grading System"
    echo "3. Exit"
    echo "Choose an option:"
    read choice

    case $choice in
        1) arithmetic_operations ;;
        2) grading_system ;;
```

```
    3) echo "Exiting..."; exit 0 ;;
    *) echo "Invalid choice. Please try again." ;;
  esac
done
```

```
Arithmetic Functions and Grading System
1. Perform Arithmetic Operations
2. Grading System
3. Exit
Choose an option:
1
Enter the first number:
2
Enter the second number:
4
Choose an operation:
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Modulus
1
Result: 6
```

Write a shell script program for a number to check whether it is

1. palindrome

2. prime

3. generate Fibonacci series

4. Armstrong, no

```
#!/bin/bash
# Function to check if a number is a palindrome
is_palindrome() {
  echo "Enter a number to check if it is a palindrome:"
  read num
  original=$num
  reverse=0

  while [ $num -gt 0 ]; do
    remainder=$((num % 10))
    reverse=$((reverse * 10 + remainder))
    num=$((num / 10))
  done

  if [ $original -eq $reverse ]; then
    echo "$original is a palindrome."
```

```
    else
        echo "$original is not a palindrome."
    fi
}

# Function to check if a number is prime
is_prime() {
    echo "Enter a number to check if it is prime:"
    read num
    if [ $num -lt 2 ]; then
        echo "$num is not a prime number."
        return
    fi

    for ((i = 2; i * i <= num; i++)); do
        if [ $((num % i)) -eq 0 ]; then
            echo "$num is not a prime number."
            return
        fi
    done
    echo "$num is a prime number."
}

# Function to generate Fibonacci series
generate_fibonacci() {
    echo "Enter the number of terms for the Fibonacci series:"
    read terms
    a=0
    b=1
    echo "Fibonacci series up to $terms terms:"
    for ((i = 0; i < terms; i++)); do
```

```
        echo -n "$a "
        temp=$((a + b))
        a=$b
        b=$temp
    done
    echo
}

# Function to check if a number is an Armstrong number
is_armstrong() {
    echo "Enter a number to check if it is an Armstrong number:"
    read num
    original=$num
    sum=0

    while [ $num -gt 0 ]; do
        digit=$((num % 10))
        sum=$((sum + digit * digit * digit))
        num=$((num / 10))
    done

    if [ $sum -eq $original ]; then
        echo "$original is an Armstrong number."
    else
        echo "$original is not an Armstrong number."
    fi
}

# Main menu
while true; do
    echo ""
```

```
    echo "Number Operations Menu"
    echo "1. Check Palindrome"
    echo "2. Check Prime"
    echo "3. Generate Fibonacci Series"
    echo "4. Check Armstrong"
    echo "5. Exit"
    echo "Enter your choice:"
    read choice

    case $choice in
        1) is_palindrome ;;
        2) is_prime ;;
        3) generate_fibonacci ;;
        4) is_armstrong ;;
        5) echo "Exiting..."; exit 0 ;;
        *) echo "Invalid choice. Please try again." ;;
    esac
done
```

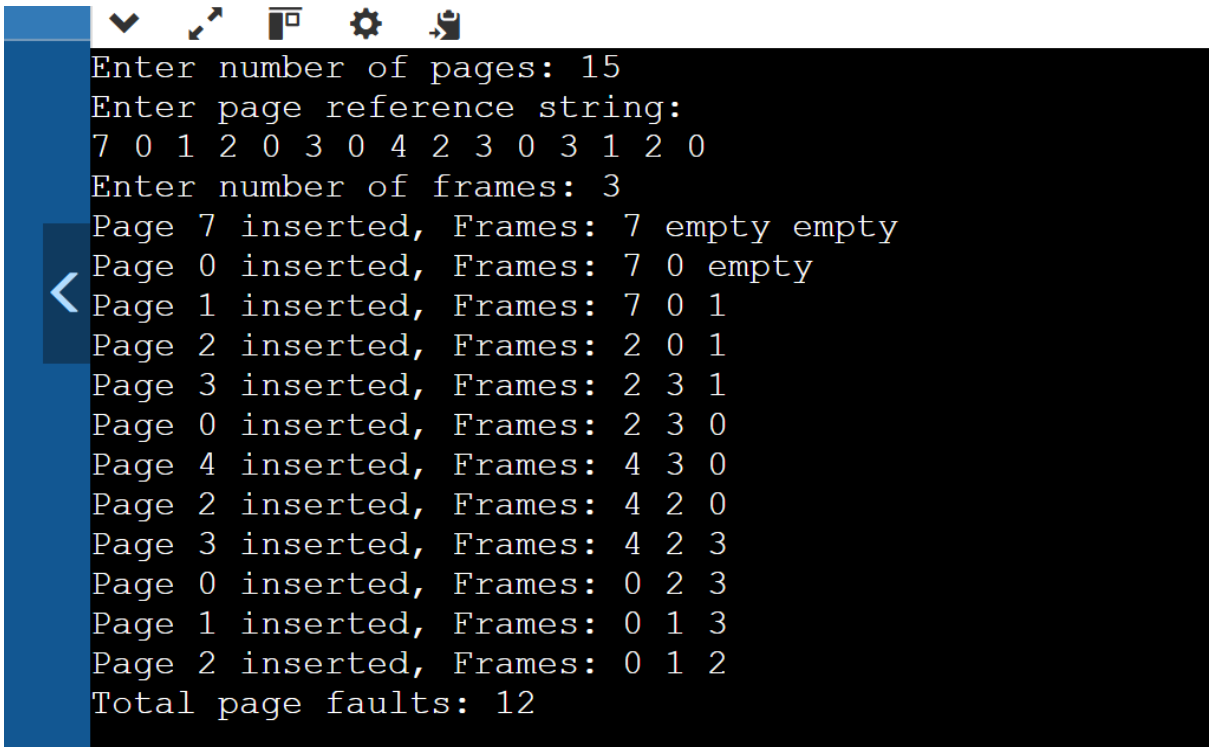
Page Replacement:

FIFO:

```
#include <stdio.h>
#define MAX_FRAMES 10
#define MAX_PAGES 20
void FIFO(int pages[], int numPages, int numFrames) {
    int frames[MAX_FRAMES];
    int pageFaults = 0;
    int nextFrame = 0;
       // Initialize frames to -1, meaning empty
    for (int i = 0; i < numFrames; i++) {
```

```
        frames[i] = -1;
    }
   // Iterate over the page reference string
    for (int i = 0; i < numPages; i++) {
        int page = pages[i];
        int pageFound = 0;
  // Check if the page is already in the frame (hit)
        for (int j = 0; j < numFrames; j++) {
            if (frames[j] == page) {
                pageFound = 1;
                break;
            }
        }
       // If the page was not found in the frames, it's a page fault
        if (!pageFound) {
            // Replace the oldest page (FIFO)
            frames[nextFrame] = page;
            nextFrame = (nextFrame + 1) % numFrames;
            pageFaults++;
 // Print the current state of the frames after replacement
            printf("Page %d inserted, Frames: ", page);
            for (int j = 0; j < numFrames; j++) {
                if (frames[j] == -1) {
                    printf("empty ");
                } else {
                    printf("%d ", frames[j]);
                }
            }
            printf("\n");
        }
    }
```

```
    printf("Total page faults: %d\n", pageFaults);
}
int main() {
    int pages[MAX_PAGES];
    int numPages, numFrames;
   printf("Enter number of pages: ");
    scanf("%d", &numPages);
printf("Enter page reference string:\n");
    for (int i = 0; i < numPages; i++) {
        scanf("%d", &pages[i]);
    }
 printf("Enter number of frames: ");
    scanf("%d", &numFrames);
  FIFO(pages, numPages, numFrames);
 return 0;
}
```



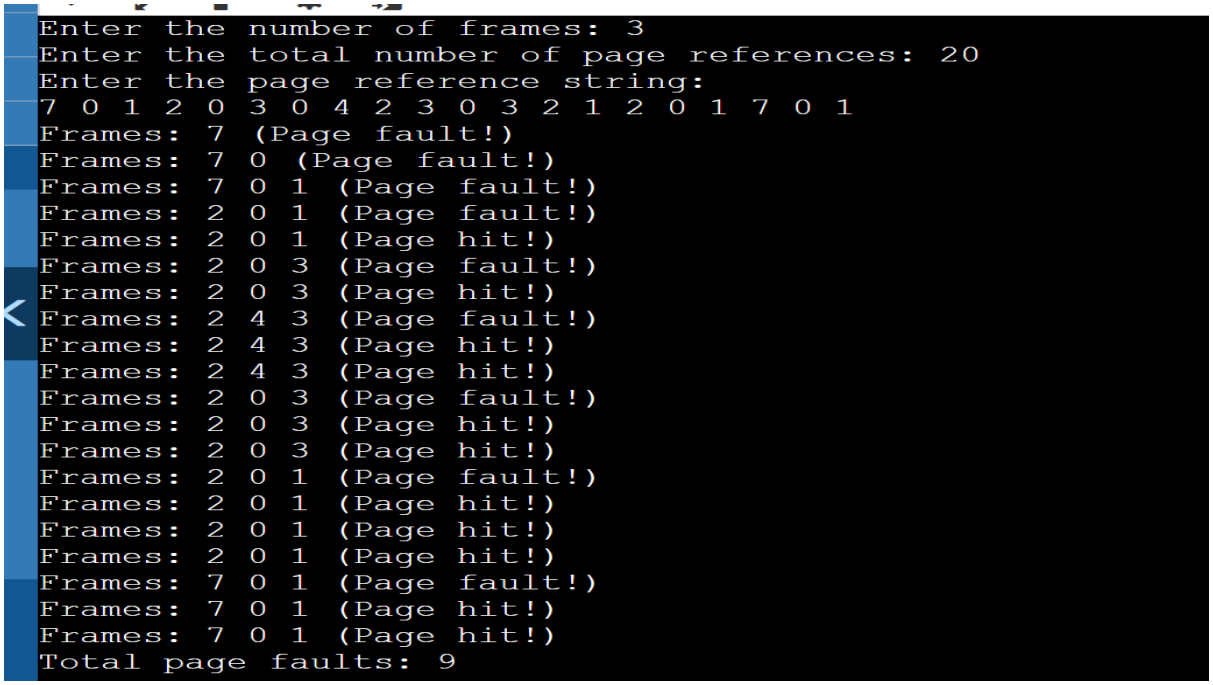
Optimal Page replacement:

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

#define MAX_FRAMES 10
#define MAX_PAGES 100

// Function to find the index of the page that will not be used for the longest time in future
int findOptimal(int frames[], int n, int page[], int currPos, int totalPages) {
    int farthest = currPos;
    int indexToReplace = -1;

    // Check all pages in memory
    for (int i = 0; i < n; i++) {
        int j;
        for (j = currPos; j < totalPages; j++) {
            if (frames[i] == page[j]) {
                if (j > farthest) {
                    farthest = j;
                    indexToReplace = i;
                }
                break;
            }
        }
        // If page is not found in future, it will be replaced
        if (j == totalPages) {
            return i;
        }
    }

    return indexToReplace;
```

```
}

// Function to implement Optimal page replacement
void optimalPageReplacement(int page[], int totalPages, int n) {
    int frames[MAX_FRAMES];
    int pageFaults = 0;
    int hit = 0;

    // Initialize frames as empty
    for (int i = 0; i < n; i++) {
        frames[i] = -1;
    }

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

        // Check if page is already in frames
        for (int j = 0; j < n; j++) {
            if (frames[j] == page[i]) {
                found = 1;
                hit = 1;
                break;
            }
        }

        // If page is not in frames, replace a page
        if (!found) {
            // Find the index to replace using the Optimal strategy
            int replaceIndex = findOptimal(frames, n, page, i + 1, totalPages);
            frames[replaceIndex] = page[i];
            pageFaults++;
```

```
        }
    printf("Frames: ");
        for (int j = 0; j < n; j++) {
            if (frames[j] != -1)
                printf("%d ", frames[j]);
        }
        if (!hit) {
            printf("(Page fault!)\n");
        } else {
            printf("(Page hit!)\n");
        }
        hit = 0;
    }

    printf("Total page faults: %d\n", pageFaults);
}

int main() {
    int page[MAX_PAGES], totalPages, n;

    // Take the input from the user for page reference string and the number of frames
    printf("Enter the number of frames: ");
    scanf("%d", &n);

    printf("Enter the total number of page references: ");
    scanf("%d", &totalPages);

    printf("Enter the page reference string:\n");
    for (int i = 0; i < totalPages; i++) {
        scanf("%d", &page[i]);
    }
```

```
    // Call the Optimal Page Replacement function
    optimalPageReplacement(page, totalPages, n);

    return 0;
}
```



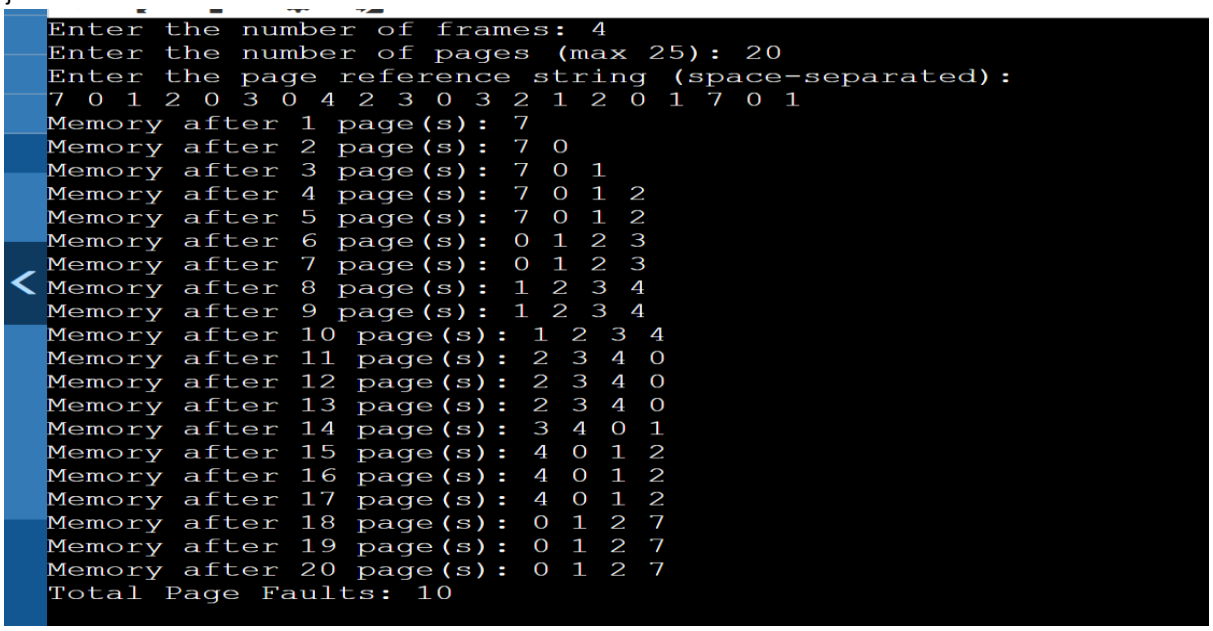
LRU:

```
#include <stdio.h>

#define MAX_FRAMES 10
#define MAX_PAGES 25  // Define the maximum number of pages

// Function to simulate LRU page replacement
void lruPageReplacement(int pages[], int numPages, int numFrames) {
    int memory[MAX_FRAMES];   // To store frames in memory
    int pageFaults = 0;
    int i, j, k;

    // Initialize memory with -1 (empty frames)
```

```
for (i = 0; i < numFrames; i++) {
    memory[i] = -1;
}

// Iterate through the page reference string
for (i = 0; i < numPages; i++) {
    int page = pages[i];
    int found = 0;

    // Check if the page is already in memory (no page fault)
    for (j = 0; j < numFrames; j++) {
        if (memory[j] == page) {
            found = 1;
            break;
        }
    }

    // If page is not in memory (page fault)
    if (!found) {
        pageFaults++;

        // If memory is full, remove the least recently used page
        for (k = 0; k < numFrames - 1; k++) {
            memory[k] = memory[k + 1];
        }

        // Insert the new page at the last position in memory
        memory[numFrames - 1] = page;
    }

    // Display the current memory content after every page reference
```

```
        printf("Memory after %d page(s): ", i + 1);
        for (j = 0; j < numFrames; j++) {
            if (memory[j] != -1) {
                printf("%d ", memory[j]);
            }
        }
        printf("\n");
    }

    // Print the total page faults
    printf("Total Page Faults: %d\n", pageFaults);
}

int main() {
    int numPages, numFrames;

    // Take input from the user for number of frames
    printf("Enter the number of frames: ");
    scanf("%d", &numFrames);
    if (numFrames > MAX_FRAMES) {
        printf("Max frames allowed are %d. Setting frames to %d.\n", MAX_FRAMES, MAX_FRAMES);
        numFrames = MAX_FRAMES;
    }

    // Take input for the page reference string
    printf("Enter the number of pages (max %d): ", MAX_PAGES);
    scanf("%d", &numPages);
    if (numPages > MAX_PAGES) {
        printf("Max pages allowed are %d. Setting pages to %d.\n", MAX_PAGES, MAX_PAGES);
        numPages = MAX_PAGES;
    }
```

```
    int pages[numPages];

    printf("Enter the page reference string (space-separated):\n");
    for (int i = 0; i < numPages; i++) {
        scanf("%d", &pages[i]);
    }

    // Call the LRU page replacement function
    lruPageReplacement(pages, numPages, numFrames);

    return 0;
}
```



Clock replacement:

```
#include <stdio.h>

#define MAX_FRAMES 10
#define MAX_PAGES 25

void clockPageReplacement(int pages[], int numPages, int numFrames) {
    int frames[numFrames];   // To store pages in memory
```

```
int refBit[numFrames];    // To keep track of reference bits
int hand = 0;             // Points to the current page in the clock
int pageFaults = 0;
int i, j;

// Initialize memory frames and reference bits to -1 and 0 respectively
for (i = 0; i < numFrames; i++) {
   frames[i] = -1;
   refBit[i] = 0;
}

// Iterate through the page reference string
for (i = 0; i < numPages; i++) {
   int page = pages[i];
   int found = 0;

   // Check if the page is already in one of the frames
   for (j = 0; j < numFrames; j++) {
      if (frames[j] == page) {
         found = 1;
         refBit[j] = 1;  // Set the reference bit to 1 (used recently)
         break;
      }
   }

   // If the page is not in memory (page fault)
   if (!found) {
      pageFaults++;

      // Find a page to replace using the clock algorithm
      while (refBit[hand] == 1) {
```

```
            refBit[hand] = 0;   // Reset reference bit to 0 for pages that were used
            hand = (hand + 1) % numFrames;   // Move to the next page (circular)
        }

        // Replace the page at the clock hand position
        frames[hand] = page;
        refBit[hand] = 1;  // Set the reference bit to 1 for the newly added page
        hand = (hand + 1) % numFrames;  // Move the clock hand to the next position
    }

    // Display the current memory content after every page reference
    printf("Memory after %d page(s): ", i + 1);
    for (j = 0; j < numFrames; j++) {
        if (frames[j] != -1) {
            printf("%d ", frames[j]);
        }
    }
    printf("\n");
  }

  // Print the total page faults
  printf("Total Page Faults: %d\n", pageFaults);
}

int main() {
  int numPages, numFrames;

  // Take input from the user for number of frames
  printf("Enter the number of frames: ");
  scanf("%d", &numFrames);
  if (numFrames > MAX_FRAMES) {
```

```
        printf("Max frames allowed are %d. Setting frames to %d.\n", MAX_FRAMES, MAX_FRAMES);
        numFrames = MAX_FRAMES;
    }

    // Take input for the page reference string
    printf("Enter the number of pages (max %d): ", MAX_PAGES);
    scanf("%d", &numPages);
    if (numPages > MAX_PAGES) {
        printf("Max pages allowed are %d. Setting pages to %d.\n", MAX_PAGES, MAX_PAGES);
        numPages = MAX_PAGES;
    }
  int pages[numPages];
printf("Enter the page reference string (space-separated):\n");
    for (int i = 0; i < numPages; i++) {
        scanf("%d", &pages[i]);
    }
// Call the Clock page replacement function
    clockPageReplacement(pages, numPages, numFrames);

    return 0;
}
```

```
Enter the number of frames: 4
Enter the number of pages (max 25): 20
Enter the page reference string (space-separated):
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Memory after 1 page(s): 7
Memory after 2 page(s): 7 0
Memory after 3 page(s): 7 0 1
Memory after 4 page(s): 7 0 1 2
Memory after 5 page(s): 7 0 1 2
Memory after 6 page(s): 3 0 1 2
Memory after 7 page(s): 3 0 1 2
Memory after 8 page(s): 3 0 4 2
Memory after 9 page(s): 3 0 4 2
Memory after 10 page(s): 3 0 4 2
Memory after 11 page(s): 3 0 4 2
Memory after 12 page(s): 3 0 4 2
Memory after 13 page(s): 3 0 4 2
Memory after 14 page(s): 3 0 4 1
Memory after 15 page(s): 2 0 4 1
Memory after 16 page(s): 2 0 4 1
Memory after 17 page(s): 2 0 4 1
Memory after 18 page(s): 2 0 7 1
Memory after 19 page(s): 2 0 7 1
Memory after 20 page(s): 2 0 7 1
Total Page Faults: 9
```

Phase 1:

```
#include <algorithm>
#include <fstream>
#include <iostream>
#include <string>
using namespace std;
class VM{
private:
   char buffer[40];
   char Memory[100][4];
   char IR[4];
   char R[4];
   bool C;
   int IC;
   int SI;
   fstream infile;
   ofstream outfile;
   void resetAll(){
      fill(buffer, buffer + sizeof(buffer), '\0');
      fill(&Memory[0][0], &Memory[0][0] + sizeof(Memory), '\0');
      fill(IR, IR + sizeof(IR), '\0');
      fill(R, R + sizeof(R), '\0');
      IC = 0;
      C = true;
      SI = 0;
   }
void resetBuffer(){
      fill(buffer, buffer + sizeof(buffer), '\0');
   }  void masterMode()
   {
      switch (SI)
```

```
    {
    case 1:
       READ();
       break;
 case 2:
       WRITE();
       break;
case 3:
       TERMINATE();
       break;
    }
    SI = 0;
  }
void READ()
  {
    cout << "Read function called\n";
    string data;
    getline(infile, data);
    int len = data.size();
    for (int i = 0; i < len; i++)
    {
       buffer[i] = data[i];
    }
    int buff = 0, mem_ptr = (IR[2] - '0') * 10;
    while (buff < 40 && buffer[buff] != '\0')
    {
       for (int i = 0; i < 4; i++)
       {
          Memory[mem_ptr][i] = buffer[buff];
          buff++;
       }
```

```
        mem_ptr++;
    }
    resetBuffer();
}

void WRITE()
{
    cout << "Write function called\n";
    outfile.open("output.txt", ios::app);

    for (int i = (IR[2] - '0') * 10; i < (IR[2] - '0' + 1) * 10; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (Memory[i][j] != '\0')
            {
                outfile << Memory[i][j];
            }
        }
    }
    outfile << "\n";
    outfile.close();
}

void TERMINATE()
{
    outfile.open("output.txt", ios::app);
    cout << "Terminate called\n\n";
    outfile << "\n\n";
    outfile.close();
}
```

```
void LOAD(){

   if (infile.is_open())
   {
      string s;
      while (getline(infile, s))
      {
         if (s[0] == '$' && s[1] == 'A' && s[2] == 'M' && s[3] == 'J'){
            resetAll();
            cout<<"New Job started\n";
         }
         else if (s[0] == '$' && s[1] == 'D' && s[2] == 'T' && s[3] == 'A'){
            cout << "Data card loding\n";
            resetBuffer();
            MOSstartexe();
         }

         else if (s[0] == '$' && s[1] == 'E' && s[2] == 'N' && s[3] == 'D'){
            cout << "END of Job\n";

            }

         else{

            cout << "Program Card loding\n";
            int length = s.size();
            resetBuffer();

            for (int i = 0; i < length; i++){
               buffer[i] = s[i];
```

```
            }
            int buff = 0;
            int ref = 0;

            while (buff < 40 && buffer[buff] != '\0'){
               for (int j = 0; j < 4; j++){
                  if (buffer[buff] == 'H'){
                     Memory[IC][j] = 'H';
                     buff++;
                     break;
                  }
                  Memory[IC][j] = buffer[buff];
                  buff++;
               }
               IC++;
            }
         }
      }
      infile.close();
   }
}

void MOSstartexe(){
   IC = 0;
   executeUserProgram();
}

void executeUserProgram(){
   while (IC < 99 && Memory[IC][0] != '\0'){

      for (int i = 0; i < 4; i++){
```

```
    IR[i] = Memory[IC][i];
}

IC++;

if (IR[0] == 'G' && IR[1] == 'D'){
    SI = 1;
    masterMode();
}

else if (IR[0] == 'P' && IR[1] == 'D'){
    SI = 2;
    masterMode();
}

else if (IR[0] == 'H'){
    SI = 3;
    masterMode();
    return;
}

else if (IR[0] == 'L' && IR[1] == 'R'){
    for (int i = 0; i < 4; i++){
        R[i] = Memory[(IR[2] - '0') * 10 + (IR[3] - '0')][i];
    }
}

else if (IR[0] == 'S' && IR[1] == 'R'){
    for (int i = 0; i < 4; i++){
        Memory[(IR[2] - '0') * 10 + (IR[3] - '0')][i] = R[i];
    }
```

```
        }

        else if (IR[0] == 'C' && IR[1] == 'R'){
            int cnt = 0;
            for (int i = 0; i < 4; i++  ){
                if (Memory[(IR[2] - '0') * 10 + (IR[3] - '0')][i] == R[i]){
                    cnt++;
                }
            }
            if (cnt == 4){
                C = true;
            }
            else{
                C = false;
            }
        }

        else if (IR[0] == 'B' && IR[1] == 'T'){
            if (C){
                IC = (IR[2] - '0') * 10 + (IR[3] - '0');
            }
        }
    }
}

public:
    VM()
    {
        infile.open("input.txt", ios::in);
        resetAll();
        LOAD();
```

```
    }
};

int main()
{
    VM v;
    return 0;
}
```

Input file:

```
$AMJ000100030001
GD10PD10H
$DTA
Hello World!
$END0001
$AMJ000200110004
GD20GD30GD40GD50LR20CR30BT09PD50H
PD40H
$DTA
VIT
VIIT
IS SAME
IS NOT SAME
$END0002
$AMJ0003000150001
GD20LR20SR30SR31SR32SR40SR42PD30PD40
PD40PD40PD30H
$DTA
$
$END0003
$AMJ0004000120004
```

GD20LR20SR35PD30SR43SR47PD40SR51SR55

SR59PD50H

$DTA

*

$END0004

$AMJ000500070002

GD20LR26CR20BT06GD30PD30PD20H

$DTA

RAM IS OLDER THAN SHRIRAM

NOT IN EXISTANCE

$END0005

Phase 2:

```
/*Phase2*/
#include <algorithm>
#include <fstream>
#include <iostream>
#include <string>
#include <cstdlib>
#include <time.h>
using namespace std;
class PCB{
   public:
      int jobID, TTL, TLL;
      int TTC,LLC;
};

class VM
{
private:
   fstream infile;
   ofstream outfile;
   char Memory[300][4], buffer[40], IR[4], R[4];
   bool C;
   int IC;
```

```
int SI, PI, TI;
PCB pcb;
int PTR;
int PTE;
int RA,VA;
int pageTable[30];
int pageTablePTR;
int page_fault_valid = 0;
bool Terminate;
int pageNo;

void init()
{
   fill(buffer, buffer + sizeof(buffer), '\0');
   fill(&Memory[0][0], &Memory[0][0] + sizeof(Memory), '\0');
   fill(IR, IR + sizeof(IR), '\0');
   fill(R, R + sizeof(R), '\0');
   C = true;
   IC = 0;
   SI = PI = TI= 0;

   pcb.jobID = pcb.TLL = pcb.TTL = pcb.TTC = pcb.LLC = 0;
   PTR=PTE=pageNo=-1;
   fill(pageTable,pageTable + sizeof(pageTable), 0);
   pageTablePTR = 0;
   Terminate=false;
}

void restBuffer()
{
   fill(buffer, buffer + sizeof(buffer), '\0');
}

int Allocate(){
   int pageNo;
   bool check=true;
   while(check){
      pageNo = (rand() % 30) ;
```

```
        if(pageTable[pageNo]==0){
            pageTable[pageNo] = 1;
            check=false;
        }
    }

    return pageNo;
}

void MOS()
{


    if(TI == 0 && SI == 1) {
        READ(); }
    else if(TI == 0 && SI == 2) {
        WRITE(); }
    else if(TI == 0 && SI == 3) {
        TERMINATE(0); }
    else if(TI == 2 && SI == 1) {
        TERMINATE(3); }
    else if(TI == 2 && SI == 2) {
        WRITE();
        TERMINATE(3); }
    else if (TI == 2 && SI == 3) {
        TERMINATE(0); }



    else if (TI == 0 && PI == 1)
    {
        TERMINATE(4);
    }
    else if(TI == 0 && PI == 2) {
        TERMINATE(5); }
    else if (TI == 0 && PI == 3){

        if (page_fault_valid == 1)
        {
```

```
            cout << "Valid Page Fault:   ";
            pageNo = Allocate();
            Memory[PTE][0] = (pageNo / 10) + '0';
            Memory[PTE][1] = (pageNo % 10) + '0';
            pageTablePTR++;
            PI = 0;
            cout<<"Allocated Page Number: "<<pageNo<<"\n";
        }
        else
        {
            TERMINATE(6);
        }
    }
    else if(TI == 2 && PI == 1) {
        TERMINATE(3);
    }
    else if(TI == 2 && PI == 2) {
        TERMINATE(3);
    }
    else if(TI == 2 && PI == 3) {
        TERMINATE(3); }

}

void READ()
{
    cout << "Read function called\n";

    string data;
    getline(infile, data);
    if (data[0] == '$' && data[1] == 'E' && data[2] == 'N' && data[3] == 'D'){
        TERMINATE(1);
        return;
    }
    int len = data.size();
    for (int i = 0; i < len; i++)
    {
        buffer[i] = data[i];
```

```
    }

    int buff = 0, mem_ptr = RA , end = RA + 10 ;
    while (buff < 40 && buffer[buff] != '\0' && mem_ptr < end)
    {
        for (int i = 0; i < 4; i++)
        {
            Memory[mem_ptr][i] = buffer[buff];
            buff++;
        }
        mem_ptr++;
    }
    restBuffer();
    SI=0;
}

void WRITE()
{
    cout << "Write function called\n";

    pcb.LLC++;
    if (pcb.LLC > pcb.TLL)
    {
        TERMINATE(2);
        return;
    }

    outfile.open("output.txt", ios::app);
    bool flag=true;
    for (int i = RA; i < RA + 10; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (Memory[i][j] != '\0')
            {
                outfile << Memory[i][j];
            }
            else{
```

```
            flag=false;
            break;
          }
        }
        if(!flag){
          break;
        }
    }
    SI=0;
    outfile << "\n";
    outfile.close();
}

void TERMINATE(int EM)
{
    Terminate=true;
    outfile.open("output.txt", ios::app);
    outfile << "\n";
    switch (EM)
    {
    case 0:
        outfile << "No Error: Program executed successfully\n";
        break;
    case 1:
        outfile << "Error: Out of Data\n";
        break;
    case 2:
        outfile << "Error: Line Limit Exceeded\n";
        break;
    case 3:

        if (TI == 2 && PI == 1){
            outfile << "Error: Operation Code Error\n";
        }
        if (TI == 2 && PI == 2){
            outfile << "Error: Operand Error\n";
        }
        outfile << "Error: Time Limit Exceeded\n";
```

```
        break;
    case 4:
        outfile << "Error: Operation Code Error\n";
        break;
    case 5:
        outfile << "Error: Operand Error\n";
        break;
    case 6:
        outfile << "Error: Invalid Page Fault\n";
    }
    outfile << "Job Id :" << pcb.jobID << "  ";
    outfile << "IC: " << IC<<"  ";
    outfile << "IR: ";
    for (int i = 0; i < 4; i++){
        if (IR[i] != '\0')
            outfile << IR[i];
    }
    outfile << "  ";
    outfile << "SI: " << SI << "  ";
    outfile << "PI: " << PI << "  ";
    outfile << "TI: " << TI << "  ";
    outfile << "TLL: " << pcb.TLL << "  ";
    outfile << "LLC: " << pcb.LLC << "  ";
    outfile << "TTL: " << pcb.TTL << "  ";
    outfile << "TTC: " << pcb.TTC << "  ";

    outfile << "\n\n\n";
    SI = 0;
    PI = 0;
    TI = 0;

    outfile.close();
}

void LOAD()
{
```

```
if (infile.is_open())
{
   string s;
   while (getline(infile, s))
   {
      if (s[0] == '$' && s[1] == 'A' && s[2] == 'M' && s[3] == 'J')
      {

         init();
         cout << "New Job started\n";
         pcb.jobID = (s[4] - '0') * 1000 + (s[5] - '0') * 100 + (s[6] - '0') * 10 + (s[7] - '0');
         pcb.TTL = (s[8] - '0') * 1000 + (s[9] - '0') * 100 + (s[10] - '0') * 10 + (s[11] - '0');
         pcb.TLL = (s[12] - '0') * 1000 + (s[13] - '0') * 100 + (s[14] - '0') * 10 + (s[15] - '0');




         PTR = Allocate()*10;
         for(int i=PTR;i<PTR+10;i++){
            for(int j=0;j<4;j++){
               Memory[i][j]='*';
            }
         }
         cout << "\nAllocated Page is for Page Table: " << PTR / 10 << "\n";
         cout << "jobID: " << pcb.jobID << "\nTTL: " << pcb.TTL << "\nTLL: " << pcb.TLL << "\n";



      }

      else if (s[0] == '$' && s[1] == 'D' && s[2] == 'T' && s[3] == 'A')
      {
         cout << "Data card loding\n";
         restBuffer();
         MOSstartexe();
      }

      else if (s[0] == '$' && s[1] == 'E' && s[2] == 'N' && s[3] == 'D')
      {
         cout << "END of Job\n";
```

```
}

else
{
   restBuffer();

   pageNo = Allocate();
   Memory[PTR + pageTablePTR][0] = (pageNo / 10) + '0';
   Memory[PTR + pageTablePTR][1] = (pageNo % 10) + '0';
   pageTablePTR++;

   cout << "Program Card loding\n";

   int length = s.size();



   for (int i = 0; i < length; i++)
   {
      buffer[i] = s[i];
   }

   int buff = 0;
   IC = pageNo * 10;
   int end = IC + 10;



   while (buff < 40 && buffer[buff] != '\0' && IC < end)
   {
      for (int j = 0; j < 4; j++)
      {
         if (buffer[buff] == 'H')
         {
            Memory[IC][j] = 'H';
            buff++;
            break;
         }
```

```
                Memory[IC][j] = buffer[buff];
                buff++;
              }
              IC++;
            }
          }
        }
        infile.close();
    }
}

int ADDRESSMAP(int VA)
{
    if (0 <= VA  &&  VA < 100)
    {
        PTE = PTR + (VA / 10);
        if (Memory[PTE][0] == '*')
        {
            PI = 3;
            MOS();
        }
        else
        {
            string p;
            p = Memory[PTE][0];
            p += Memory[PTE][1];
            int pageNo = stoi(p);
            RA = pageNo * 10 + (VA % 10);
            return RA;
        }
    }
    else
    {
        PI = 2;
        MOS();
    }
    return pageNo * 10;
}
```

```
void MOSstartexe()
{
    IC = 0;
    executeUserProgram();
}


void executeUserProgram()
{
    while (!Terminate)
    {
        RA = ADDRESSMAP(IC);
        if(PI != 0){
            return;
        }

        for (int i = 0; i < 4; i++){
            IR[i] = Memory[RA][i];
        }

        IC++;

        string op;
        op += IR[2];
        op += IR[3];



        if (IR[0] == 'G' && IR[1] == 'D')
        {
            SIMULATION();
            page_fault_valid = 1;
            if (!isdigit(IR[2]) || !isdigit(IR[3]))
            {
                PI = 2;
                MOS();
            }
            else{
```

```
        VA = stoi(op);
        RA = ADDRESSMAP(VA);

        SI = 1;
        MOS();
    }
}


else if (IR[0] == 'P' && IR[1] == 'D')
{
    SIMULATION();
    page_fault_valid=0;
    if (!isdigit(IR[2]) || !isdigit(IR[3]))
    {
        PI = 2;
        MOS();
    }
    else{
        VA = stoi(op);
        RA = ADDRESSMAP(VA);
        SI = 2;
        MOS();
    }
}


else if (IR[0] == 'H' && IR[1] == '\0')
{
    SIMULATION();
    SI = 3;
    MOS();
    return;
}


else if (IR[0] == 'L' && IR[1] == 'R')
{
```

```
    SIMULATION();
    page_fault_valid = 0;
    if (!isdigit(IR[2]) || !isdigit(IR[3]))
    {
        PI = 2;
        MOS();
    }
    else{
        VA = stoi(op);
        RA = ADDRESSMAP(VA);
        for (int i = 0; i < 4; i++)
        {
            R[i] = Memory[RA][i];
        }
    }
}



else if (IR[0] == 'S' && IR[1] == 'R')
{
    SIMULATION();
    page_fault_valid = 1;
    if (!isdigit(IR[2]) || !isdigit(IR[3]))
    {
        PI = 2;
        MOS();
    }
    else{
        VA = stoi(op);
        RA = ADDRESSMAP(VA);
        for (int i = 0; i < 4; i++)
        {
            Memory[RA][i] = R[i];
        }
    }
}
```

```
else if (IR[0] == 'C' && IR[1] == 'R')
{
   SIMULATION();
   page_fault_valid=0;
   if (!isdigit(IR[2]) || !isdigit(IR[3]))
   {
      PI = 2;
      MOS();
   }
   else{
      VA = stoi(op);
      RA = ADDRESSMAP(VA);
      string s1,s2;
      for (int i = 0; i < 4; i++)
      {
         s1+=Memory[RA][i];
         s2+=R[i];
      }
      if (s1 == s2)
      {
         C = true;
      }
      else
      {
         C = false;
      }
   }
}



else if (IR[0] == 'B' && IR[1] == 'T')
{
   SIMULATION();
   page_fault_valid=0;
   if (!isdigit(IR[2]) || !isdigit(IR[3]))
   {
      PI = 2;
      MOS();
```

```
            }
            else{
            if (C)
              {
                 string j;
                 j+=IR[2];
                 j+=IR[3];
                 IC = stoi(j);
              }
            }
          }
          else{
            PI = 1;
            SI = 0;
            MOS();

          }
       }
    }

void SIMULATION(){
    if (IR[0] == 'G' && IR[1] == 'D'){
       pcb.TTC += 2;
    }

    else if (IR[0] == 'P' && IR[1] == 'D'){
       pcb.TTC += 1;
    }

    else if (IR[0] == 'H'){
       pcb.TTC += 1;
    }

    else if (IR[0] == 'L' && IR[1] == 'R'){
       pcb.TTC += 1;
    }

    else if (IR[0] == 'S' && IR[1] == 'R'){
```

```
            pcb.TTC += 2;
        }

        else if (IR[0] == 'C' && IR[1] == 'R'){
            pcb.TTC += 1;
        }

        else if (IR[0] == 'B' && IR[1] == 'T'){
            pcb.TTC += 1;
        }

        if(pcb.TTC >= pcb.TTL){
            TI=2;
            MOS();
        }
    }

public:
    VM()
    {
        infile.open("input.txt", ios::in);
        init();
        LOAD();
    }
};

int main()
{
    VM vm;
    return 0;

}
```