Write a shell script to generate a multiplication table

```
echo Multiplication Table:
echo Which table do you want ? (Give Number):
read num
iter=1
while [ $num –le 5 ]
do
res=`expr $num \* $iter`
echo $num “*” $iter “=” $res
iter=`expr $iter + 1`
done
```

Write a shell script that copies multiple files to a directory.

```
 iter=1
echo Enter new dir:
read nn
mkdir $nn
echo Enter number of files:
read na
while [  $iter –le $na ]
do
echo Enter file name:
read fn
cp $fn $nn
iter=`expr $iter + 1`
```

done

Write a shell script which counts the number of lines and words present in a given file.

```
echo Enter a file name:
read fn
echo Number of Lines:
wc –l $fn
echo Number of Words:
wc –w $fn
echo Number of Characters:
wc –c $fn
```

Write a shell script which displays the list of all files in the given directory.

```
echo Menu
echo 1.Short format display
echo 2.Long format display
echo 3.Hidden files to display
echo Enter ur choice:
read ch
case ch in
1)      ls $a;;
2)      ls –l $a;;
3)      ls –la $a;;
*)  echo Choice is not correct;;
Esac
```

Write a shell script(small calculator) that adds, subtracts, multiplies and divides the given two integers. There are two division options: one returns the quotient and the other returns reminder. The script requires 3 arguments: The operation to be used and two integer numbers. The options are add(-a),subtract(-s), multiply(-m), quotient(-c) and reminder(-r).

```
echo "Enter First Value "

read x

echo "Enter Second Value "

read y

while  [$q –ne 0 ]

do

echo “Enter –a for adding”

echo “Enter –s for subtraction”

echo “Enter –m for multiplication”

echo “Enter –c for Quotient”

echo “Enter –r for reminder”

read s

case $s in

-a)  p=`expr $x + $y`

    Echo "Sum = $p"

;;

-b)  p=`expr $x - $y`

    Echo "difference  = $p"

;;

-m)  p=`expr $x \* $y`

    Echo "Product = $p"

;;
```

```
-c)  p=`expr $x / $y`
    Echo "quotient = $p"

;;

-r)  p=`expr $x % $y`
    Echo “reminder = $p"
;;
```

Write a shell script to reverse the rows and columns of a matrix.

```
Echo "Enter Number of rows"
read  r
Echo "Enter Number of columns"
read  c
i=0
echo "Enter elements"
until [ $i –eq `expr $r \* $c` ]
do
     read a[$i]
     i= `expr $i + 1`
done
i=0 ;  k=0
echo "Transpose of a Matrix"
until [ $i –eq $c ]
do
     j=0;
```

```
    until [ $j –eq $r ]
    do
        n= `expr $j \* $c`
        m= `expr $n + $i
        b[$k] = ${a[$m]}
        echo "${b[$k]} \t"
        k= `expr $k + 1`
        j= `expr $j + 1`
    done
    i = `expr $i + 1`
    echo "\n"
done
```

Write a C program that counts the number of blanks in a text file using standard I/O

```
#include <fcntl.h>
#include < sys/stat.h>
#include <stdio.h>
int main(int argc, char **argv)
{
  FILE *fd1;
  int n,count=0;
  char buf;
  fd1=fopen(argv[1],"r");
  while(!feof(fd1))
  {
```

```
  buf=fgetc(fd1);
   if(buf==' ')
      count=count+1;
  }
  printf("\n Total Blanks= %d",count);
return (0);
}
```

Write a C program that counts the number of blanks in a text file using system calls

```
#include<fcntl.h>
#include<sys/stat.h>

int main(int argc, char **argv)
{
  int fd1;
  int n,count=0;
  char buf;
  fd1=open(argv[1],O_RDONLY);
  while((n=read(fd1,&buf,1))>0)
  {
   if(buf==' ')
      count=count+1;
```

Implement in C the following ls Unix command using system calls

```
#include <sys/types.h>

#include <sys/dir.h>

#include <sys/param.h>

#include <stdio.h>


#define FALSE 0

#define TRUE 1


extern  int alphasort();


char pathname[MAXPATHLEN];


main()   {

int count,i;

struct dirent **files;

int file_select();


if (getwd(pathname) == NULL )

{ printf("Error getting pathn");
```

```
exit(0);
}
                printf("Current Working Directory = %sn",pathname);
        count = scandir(pathname, &files, file_select, alphasort);

                if (count <= 0)
{               printf("No files in this directoryn");
                                                exit(0);
                                        }
                printf("Number of files = %dn",count);
        for (i=1;i<count+1;++i)
                printf("%s  \n",files[i-1]->d_name);

}

int file_select(struct direct   *entry)

{
if ((strcmp(entry->d_name, ".") == 0) ||(strcmp(entry->d_name, "..") == 0))
                                        return (FALSE);
else
```

```
                    return (TRUE);
        }
```

Implement in C the Unix command mv using system calls

```
#include<fcntl.h>
#include<stdio.h>
#include<unistd.h>
#include<sys/stat.h>
int main(int argc, char **argv)
{
  int fd1,fd2;
  int n,count=0;
  fd1=open(argv[1],O_RDONLY);
fd2=creat(argv[2],S_IWUSR);
rename(fd1,fd2);
unlink(argv[1]);
return (0);
}
```

Write a c program for message passing using pipes.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
```

```
int main()
{
int fd[2];
if(pipe(fd)<0)
exit(1);
if(fork())
{
close(fd[0]);
write(fd[1], “Message from Suhrit”12);
}
else
{
char buf[100];
close(fd[1]);
read(fd[0],buf,100);
printf(“Received by Students of SuhritSolutions:%s\n”,buf);
fflush(stdout);
}
exit(0);
}
```


Write a C program that illustrates the creation of child process using fork system call. One process finds sum of even series and other process finds sum of odd series

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
```

```
#include <fcntl.h>

int main()

{

int i,n,sum=0;

pid_t pid;

system(“clear”);

printf(“Enter n value:”);

scanf(“%d”,&n)

pid=fork();

if(pid==0)

{

printf(“From child process\n”);

for(i=1;i<n;i+=2)

{

printf(“%d\”,i);

sum+=i;

}

printf(“Odd sum:%d\n”,sum);

}

else

{

printf(“From process\n”);

for(i=0;i<n;i+=2)

{

printf(“%d\”,i);
```

```
sum+=i;
}
printf("Even sum:%d\n",sum);
}
}
```

Write a C program that displays the real time of a day every 60 seconds

```
#include <stdio.h>
#include <sys/time.h>
#include <sys/signal.h>

/* Declarations */
    void main();
int times_up();

void main()
{

  for (; ;)
      {
        times_up(1);
        sleep(60);

      }
}
```

```
int times_up(sig)
   int sig;
   {
     long now;
     long  time(struct tms *ptr);
      char *ctime();

      time (&now);
      printf("It is now %s\n", ctime (&now));
return (sig);
   }
```

Write a C program that illustrates file locking using semaphores.

```
#include <stdio.h>
 #include <sys/file.h>
 #include <error.h>
 #include <sys/sem.h>
 #define MAXBUF 100
 #define KEY 1216
 #define SEQFILE “suhritfile”
 int semid,fd;
 void my_lock(int);
 void my_unlock(int);
 union semnum
```

```
{
    int val;
    struct semid_ds *buf;
    short *array;
}arg;
int main(])
{
     int child, i,n, pid, seqno;
     char buff[MAXBUF+1];
     pid=getpid();
     if((semid=semget(KEY, 1, IPC_CREAT | 0666))= = -1)
     {
          perror("semget");
          exit(1);
     }
     arg.val=1;
     if(semctl(semid,0,SETVAL,arg)<0)
          perror("semctl");
     if((fd=open(SEQFILE,2))<0)
     {
          perror("open");
          exit(1);
     }
     pid=getpid();
     for(i=0;i<2;i++)
```

```
    {
        my_lock(fd);
        lseek(fd,01,0);
        if((n=read(fd,buff,MAXBUF))<0)
        {
            perror("read");
            exit(1);
        }
        printf("pid:%d, Seq no:%d\n", pid, seqno);
        seqno++;
        sprintf(buff,"%d\n", seqno);
        n=strlen(buff);
        lseek(fd,01,0);
        if(write(fd,buff,n)!=n)
        {
            perror("write");
            exit(1);
        }
        sleep(1);
        my_unlock(fd);
    }
}
  void my_lock(int fd)
  {
        struct sembuff sbuf=(0, -1, 0);
```

```
        if(semop(semid, &sbuf, 1)= =0)
            printf("Locking: Resource…\n");
        else
            printf("Error in Lock\n");
}
void my_unlock(int fd)
{
        struct sembuff sbuf=(0, 1, 0);
        if(semop(semid, &sbuf, 1)= =0)
            printf("UnLocking: Resource…\n");
        else
            printf("Error in Unlock\n");
}
```

Write a C program that implements a producer-consumer system with two processes.(using semaphores)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#define NUM_LOOPS     20
int main(int argc, char* argv[])
```

```
{
   int sem_set_id;
   union semun sem_val;
   int child_pid;
   int i;
   struct sembuf sem_op;
   int rc;
   struct timespec delay;

   sem_set_id = semget(IPC_PRIVATE, 1, 0600);
   if (sem_set_id == -1) {
            perror("main: semget");
            exit(1);
   }
   printf("semaphore set created,
 semaphore set id '%d'.\n", sem_set_id);

   sem_val.val = 0;
   rc = semctl(sem_set_id, 0, SETVAL, sem_val);
   child_pid = fork();
   switch (child_pid) {
            case -1:
               perror("fork");
               exit(1);
            case 0:
```

```
    for (i=0; i<NUM_LOOPS; i++) {
            sem_op.sem_num = 0;
            sem_op.sem_op = -1;
            sem_op.sem_flg = 0;
            semop(sem_set_id, &sem_op, 1);
            printf("consumer: '%d'\n", i);
            fflush(stdout);
        sleep(3);
    }
    break;
  default:
    for (i=0; i<NUM_LOOPS; i++)
{
            printf("producer: '%d'\n", i);
            fflush(stdout);
            sem_op.sem_num = 0;
            sem_op.sem_op = 1;
            sem_op.sem_flg = 0;
            semop(sem_set_id, &sem_op, 1);
            sleep(2);
            if (rand() > 3*(RAND_MAX/4))
            {
               delay.tv_sec = 0;
               delay.tv_nsec = 10;
               nanosleep(&delay, NULL);
```

```
                }
            }
            break;
    }

    return 0;
}
```

Write a C program that illustrates inter process communication using shared memory system calls.

```
#include <stdio.h>
 #include<sys/ipc.h>
 #include<sys/shm.h>
 #include<sys/types.h>
 #define SEGSIZE 100
 int main(int argc, char *argv[ ])
 {
      int shmid,cntr;
      key_t key;
      char *segptr;
      char buff[ ]="Hello world";
      key=ftok(".",'s');
      if((shmid=shmget(key, SEGSIZE, IPC_CREAT |
IPC_EXCL | 0666))= = -1)
      {
           if((shmid=shmget(key,SEGSIZE,0))= = -1)
```

```
        {
            perror("shmget");
            exit(1);
        }
    }
    else
    {
        printf("Creating a new shared memory seg \n");
        printf("SHMID:%d", shmid);
    }
    system("ipcs –m");
    if((segptr=shmat(shmid,0,0))= =(char*)-1)
{
        perror("shmat");
        exit(1);
}
printf("Writing data to shared memory…\n");
strcpy(segptr,buff);
printf("DONE\n");
printf("Reading data from shared memory…\n");
printf("DATA:-%s\n"segptr);
printf("DONE\n");
print("Removing shared memory Segment…\n");
if(shmctl(shmid,IPC_RMID,0)= = -1)
    printf("Can't Remove Shared memory Segment…\n");
```

```
    else
        printf("Removed Successfully");
}
```

Write a C program that illustrates the following.

a) Creating a message queue.

b) Writing to a message queue.

c) Reading from a message queue.

```
#include <stdio.h>
 #include <sys/ipc.h>
 #include <fcntl.h>
 #define MAX 255
      struct mesg
      {
          long type;
          char mtext[MAX];
      } *mesg;
      char buff[MAX];
 main()
{
      int mid,fd,n,count=0;;
      if((mid=msgget(1006,IPC_CREAT | 0666))<0)
      {
            printf("\n Can't create Message Q");
            exit(1);
```

```
    }
    printf("\n Queue id:%d", mid);
    mesg=(struct mesg *)malloc(sizeof(struct mesg));
    mesg ->type=6;
    fd=open("fact",O_RDONLY);
    while(read(fd,buff,25)>0)
    {
        strcpy(mesg ->mtext,buff);
        if(msgsnd(mid,mesg,strlen(mesg ->mtext),0)== -1)
            printf("\n Message Write Error");
    }

    if((mid=msgget(1006,0))<0)
    {
        printf("\n Can't create Message Q");
        exit(1);
    }
    while((n=msgrcv(mid,&mesg,MAX,6,IPC_NOWAIT))>0)
        write(1,mesg.mtext,n);
        count++;
    if((n= = -1)&(count= =0))
        printf("\n No Message Queue on Queue:%d",mid);

}
```