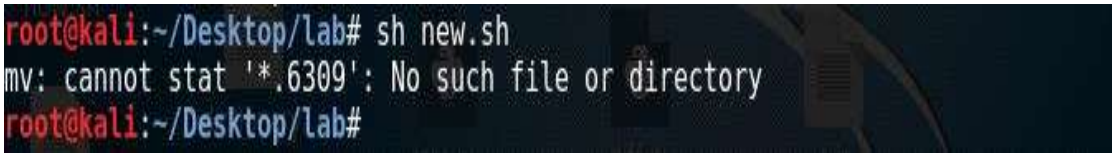
SOLUTION 1:

```
go () {
set
PS1=pwd
}
go      ##calling the function
```

SOLUTION 2:

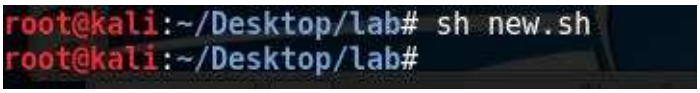
```
a=$$
for item in *.$a
do
        b=`basename –s .$a item`
        mv $item $b
done
```



SOLUTION 3:

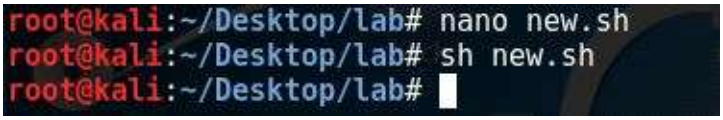
```
        mycd () {
        if [ $# -eq 1 ]
        then
                cd $1
        else
                cd ..
```

```
fi
}
mycd new            ##calling the function
mycd                ##calling the function
```



SOLUTION 4:

```
mkcd ( ) {
d=$1
mkdir –p $d
}
```



SOLUTION 5:

```
funct () {
for items in $*
do
        if [ -f $test ]
        then
                echo  “$test file existed”
        elif [ -d mycd ]
                echo “mycd exist.”
                echo “files present:” `ls /mycd | wc –l`
        else
                mkdir mycd
        fi
```

```
        done
        }
        funt f1 f2 f3            ##calling the function
```

```
root@kali:~/Desktop/lab# sh new.sh
file existed
root@kali:~/Desktop/lab#
```

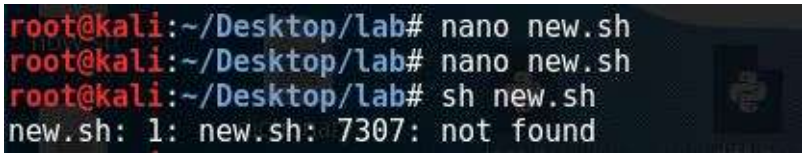
SOLUTION 6:

```
touch new.txt
touch size.txt
for items in *.*
        do
        du $items > size.txt            ##getting size in bits
        size=`cut –c 1 size.txt`
        size=`echo $size / 8 | bc`              ##converting to bytes
        if [ $size –gt 512 ]
        then
                cat $size $items >> new.txt
        fi
done
sort – rn new.txt
cut –f 2,2 new.txt
total=`wc –l new.txt`
echo “no of files $total”
```

```
root@kali:~/Desktop/lab# nano new.sh
root@kali:~/Desktop/lab# sh new.sh
4 new.txt
4 new.txt
no of files 1 new.txt
root@kali:~/Desktop/lab#
```

SOLUTION 7 :

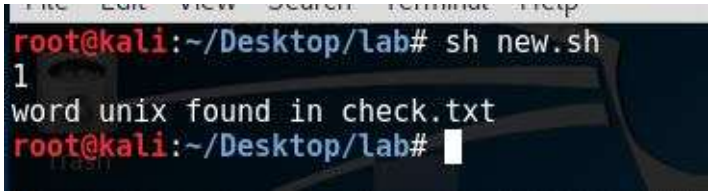
```
a=`$$`
for items in *.*
do
        if [ -f $items ]
        then
                mv $items ${items%.*}.$a
        fi
done
```



SOLUTION 8:

```
unix_search () {
        a=`grep –i “unix” $1 | wc –l`
        if [ $a –gt 0 ]
        then
                found=1
                export found            ##using found as global variable and
exporting it to
        else                            ## main shell
                found=0
        fi
        }
input_function () {
        for items in $*
        do
                unix_search $items
                if [ $found –eq 1 ]
```

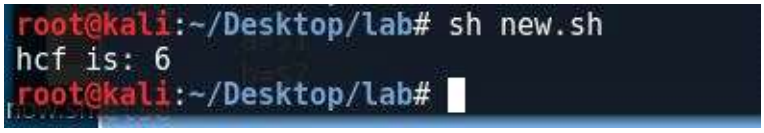
```
        then
                echo " word unix found in $items"
                exit
        fi
done
}
```



SOLUTION 9:

```
hcf () {
        a=$1
        b=$2
        if [ $1 –gt $2 ]
        then
                a=$1
                b=$2
        else
                a=$2
                b=$2
        fi
        i=$b
        while [ $i –gt 0 ]
        do
                temp1=`echo $a % $i | bc`
                temp2=`echo $b % $i | bc`
                if [ $temp1 –eq 0 –a $temp2 –eq 0 ]
                then
```

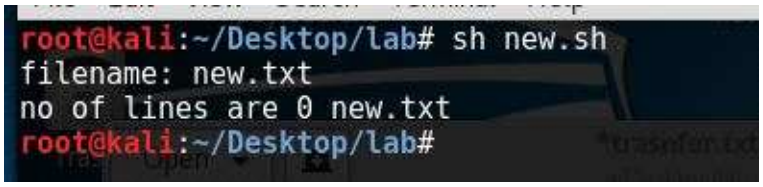
```
                result=$i
                export $result
                exit
        fi
        i=`echo $i – 1 |bc`
        done
}
lcm () {
    mult=`echo $1 * $2 | bc`
    div=hcf $1 $2
    fin=`echo $mult / $div | bc`
    result=$fin
}

hcf 12 6            ##calling the function
lcm 4 6             ##calling the function
```



SOLUTION 10 :

```
        check () {
                for items in $*
                do
        if [ -f $items ]
                then
                        echo “filename: $items”
                        lines=`wc –l $items`
                        echo “no of lines are $lines”
                else
                        echo “$items is directory”
                fi
```

```
                done
        }
check f1 dir f2.txt             ##calling the check function
```



SOLUTION 11:

```
echo “enter value of n and m”
read n m
echo “enter the file”
read file
last=`echo $n + $m |bc`
terminal=`tty`
exec > $file
i=0
while read line
do
        i=`echo $1 + 1 |bc`
        if [ $i –gt n –a $i –lt $last ]
        then
                echo $line
        fi
done
exec > $terminal
```

```
root@kali:~/Desktop/lab# nano new.sh
root@kali:~/Desktop/lab# sh new.sh
enter value of n and m
1 3
enter the file
check.txt
unix
hello
root@kali:~/Desktop/lab#
```

SOLUTION 12:

```
touch new.txt
touch size.txt
for items in *.*
        do
        du $items > size.txt            ##getting size in bits
        size=`cut –c 1 size.txt`
        size=`echo $size / 8 | bc`              ##converting to bytes
        if [ $size –gt 1000 ]
        then
                cat $size $items >> new.txt
        fi
done
sort – rn new.txt
cut –f 2,2 new.txt
total=`wc –l new.txt`
echo “no of files $total”
```

```
root@kali:~/Desktop/lab# nano new.sh
root@kali:~/Desktop/lab# sh new.sh
4 new.txt
4 new.txt
no of files 1 new.txt
root@kali:~/Desktop/lab#
```

SOLUTION 13:

```
#!/bin/bash
echo “enter values of n and m”
read n m
declare –A matrix
rows=$n
col=$m
for ((i=1;i<=rows;i++)) do
        for ((j=1;j<=col;j++)) do
                matrix[$i, $j]=$RANDOM
        done
done
f1=”%$(($${#rows}+1))s”
f2=”%9s”
printf  “$f1” ‘ ‘
for ((i=1;i<rows;i++)) do
        printf “$f2” $i
done
```

SOLUTION 14:

```
#!/bin/bash
echo “enter maximum number”
read n
echo “enter number in array”
for (( i=0;i<$n;i++))
do
        read nos[$i]
done
echo “numbers in array are:”
for ((i=0;i<$n;i++))
do
        echo ${nos[$i]}
done
for ((i=0;i<$n;i++))
```

```
do
        for(j=0;j<$n;j++))
        do
                if [ ${nos[$i]} –gt ${nos[$j]} ]
                then
                        t=${nos[$i]}
                        ${nos[$i]}= ${nos[$j]}
                        ${nos[$j]}=$t
                fi
        done
done
echo “sorted numbers”
for ((i=0;i<$n;i++))
do
        echo ${nos[$i]}
done
```

SOLUTION 15:

```
typeset -ir n=4
typeset -a Array=()
typeset -a Array1=()
typeset -a Array2=()
typeset -a Array3=()
typeset -a Array4=()
typeset -a Result=()
typeset -a Result1=()
typeset -a Result2=()


ShowMatrix() {
   typeset arr=$1
   typeset n=$2
```

```
    typeset -i i
    echo "Matrix $arr is:"
    for ((i=0;i<n;i++)) ; do
        typeset -i j
        typeset -i val
        for ((j=0;j<n;j++)) ; do
            ((val=${arr}[i*n+j]))
            printf '%5d ' $val
        done
        printf '\n';
    done
}

cut1() {
    typeset -i i
    typeset -i k=0
    for((i=0;i<n/2;i++)) ; do
        typeset -i j
        for((j=0;j<n/2;j++)); do
            ((Array1[k++] = Array[i*n+j]))
        done
    done
}

cut2() {
    typeset -i i
    typeset -i k=0
    for((i=0;i<n/2;i++)) ; do
        typeset -i j
        for((j=n/2;j<n;j++)) ; do
            ((Array2[k++] = Array[i*n+j]))
        done
    done
```

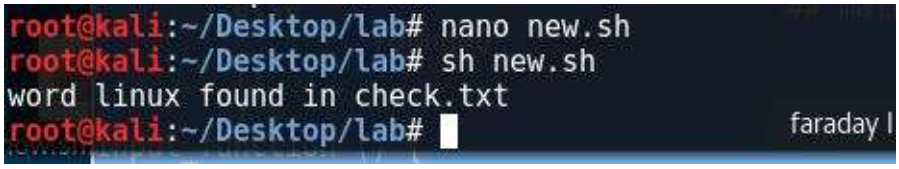
```
}

cut3() {
    typeset -i i
    typeset -i k=0
    for((i=n/2;i<n;i++)) ; do
        for((j=0;j<n/2;j++)) ; do
            ((Array3[k++] = Array[i*n+j]))
        done
    done
}

cut4() {
    typeset -i i
    typeset -i k=0
    for((i=n/2;i<n;i++)) ; do
        for((j=n/2;j<n;j++));  do
           ((Array4[k++] = Array[i*n+j]))
        done
    done
}

multiply() {
    typeset -i i
    typeset -i n=$1
    ShowMatrix Array1 $n
    ShowMatrix Array2 $n
    for((i=0;i<n;i++)); do
        typeset -i j
        for((j=0; j < n; j++)); do
            typeset -i l
            ((l=i*n+j))
            ((Result[l]=0))
```

```
            typeset -i k
            for((k=0; k<n; k++)) ; do
                    ((Result[l] += Array1[i*n+k]*Array2[k*n+j]))
            done
        done
    done
}

multiply1()
{
    typeset -i n=$1
    ShowMatrix Result $n
    ShowMatrix Array3 $n
    typeset -i i
    for((i=0; i < n; i++)) ; do
        typeset -i j
        for((j=0; j < n; j++)); do
            typeset -i l
            ((l=i*n+j))
            ((Result1[i*n+j]=0))
            typeset -i k
            for ((k=0;k<n;k++));  do
                ((Result1[l] += Result[i*n+k]*Array3[k*n+j]))
            done
        done
    done
}

multiply2() {
    typeset -i i
    typeset -i n=$1
    ShowMatrix Result1 $n
    ShowMatrix Array4 $n
```

```
    for ((i=0; i<n; i++)) ; do
        typeset -i j
        for ((j=0; j < n; j++)) ; do
            typeset -i l
            ((l=i*n+j))
            ((Result2[i*n+j]=0))
            typeset -i k
            for((k=0;k<n;k++)); do
                ((Result2[l] += Result1[i*n+k]*Array4[k*n+j]))
            done
        done
    done
}


typeset -i i
for((i=0; i<n*n; i++)) ; do
      ((Array[i]=i+1))
done

cut1
cut2
cut3
cut4

typeset -i n2
((n2 = n / 2))

multiply $n2
multiply1 $n2
multiply2 $n2
ShowMatrix Result2 $n2
```

SOLUTION 16:

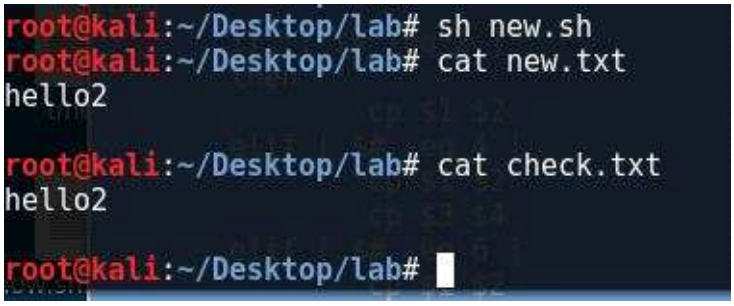
```
linux_search () {
        a=`grep –i “linux” $1 | wc –l`
        if [ $a –gt 0 ]
        then
                found=1
                export found            ##using found as global variable and exporting it to
        else                            ## main shell
                found=0
        fi
        }
input_function () {
        for items in $*         ;do
                linux_search $items
                if [ $found –eq 1 ]
                then
                        echo “ word linux found in $items”
                        exit
                fi
        done
        }
```



SOLUTION 17:

```
Check () {
        temp=`echo $# % 2 |bc`
        if [  $temp –eq 0 ]
        then
```

```
        if  [ $# -eq 2 ]
        then
                cp $1 $2
        elif [ $# -eq 4 ]
                cp $1 $2
                cp $3 $4
        elif [ $# -eq 6 ]
                cp $1 $2
                cp $3 $4
                cp $5 $6
        elif [ $# -eq 8 ]
                cp $1 $2
                cp $3 $4
                cp $5 $6
                cp $7 $8
    else
        echo “invalid no of arguments”
    fi
    }
check f1 f2 f3 f4
```



SOLUTION 18:

I:

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

```
#include <unistd.h>
#include <string.h>
#include <sys/shm.h>
#include "shm_com.h"
int main()
{
int running = 1;
void *shared_memory = (void *)0;
struct shared_use_st *shared_stuff;
int shmid;
srand((unsigned int)getpid());
shmid = shmget((key_t)1234, sizeof(struct shared_use_st), 0666 |
IPC_CREAT);
if(shmid == -1)
{
fprintf(stderr,"shmget failed\n");
exit(EXIT_FAILURE);
}
shared_memory = shmat(shmid,(void *)0,0);
if(shared_memory == (void *)-1)
{
fprintf(stderr,"shmat failed\n");
exit(EXIT_FAILURE);
}
printf("Memory attached at %X\n",(int)shared_memory);
shared_stuff = (struct shared_use_st *)shared_memory;
shared_stuff -> written_by_you = 0;
while(running)
{
if(shared_stuff -> written_by_you)
{
printf("You wrote %s",shared_stuff->some_text);
sleep(rand() % 4);
```

```
shared_stuff -> written_by_you = 0;
if(strcmp(shared_stuff->some_text,"end") == 0)
{
running = 0;
}
}
}
if(shmdt(shared_memory) == -1)
{
fprintf(stderr,"shmdt failed\n");
exit(EXIT_FAILURE);
}
if(shmctl(shmid, IPC_RMID, 0) == -1)
{
fprintf(stderr,"shmctl(IPC_RMID) Failed\n");
exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}
```

II:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/shm.h>
#include "shm_com.h"
int main()
{
int running = 1;
void *shared_memory = (void *)0;
struct shared_use_st *shared_stuff;
char buffer[BUFSIZ];
int shmid;
```

```
srand((unsigned int)getpid());
shmid = shmget((key_t)1234, sizeof(struct shared_use_st), 0666 |
IPC_CREAT);
if(shmid == -1)
{
fprintf(stderr,"shmget failed\n");
exit(EXIT_FAILURE);
}
shared_memory = shmat(shmid,(void *)0,0);
if(shared_memory == (void *)-1)
{
fprintf(stderr,"shmat failed\n");
exit(EXIT_FAILURE);
}
printf("Memory attached at %X\n",(int)shared_memory);
shared_stuff = (struct shared_use_st *)shared_memory;
while(running)
{
while(shared_stuff -> written_by_you == 1)
{
sleep(1);
printf("Waiting for client...\n");
}
printf("Enter some text... \n");
fgets(buffer,BUFSIZ,stdin);
strncpy(shared_stuff -> some_text, buffer, TEXT_SZ);
shared_stuff -> written_by_you = 1;
if(strcmp(shared_stuff -> some_text, "end") == 0)
{
running = 0;
}
}
if(shmdt(shared_memory) == -1)
```

```
{
fprintf(stderr,"shmdt failed\n");
exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}
```

SOLUTION 19:

I:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/shm.h>
#include "shm_com.h"
int main()
{
int running = 1;
void *shared_memory = (void *)0;
struct shared_use_st *shared_stuff;
int shmid;
srand((unsigned int)getpid());
shmid = shmget((key_t)1234, sizeof(struct shared_use_st), 0666 |
IPC_CREAT);
if(shmid == -1)
{
fprintf(stderr,"shmget failed\n");
exit(EXIT_FAILURE);
}
shared_memory = shmat(shmid,(void *)0,0);
if(shared_memory == (void *)-1)
{
fprintf(stderr,"shmat failed\n");
exit(EXIT_FAILURE);
```

```
}
printf("Memory attached at %X\n",(int)shared_memory);
shared_stuff = (struct shared_use_st *)shared_memory;
shared_stuff -> written_by_you = 0;
while(running)
{
if(shared_stuff -> written_by_you)
{
printf("1st Number= %d\n",shared_stuff -> aa);
printf("2nd Number= %d\n",shared_stuff -> bb);
if(shared_stuff -> aa == 0 && shared_stuff -> bb == 0)
{
running = 0;
}
printf("The product of two numbers is %d\n\n",(shared_stuff ->
aa)*(shared_stuff -> bb));
shared_stuff -> written_by_you = 0;
}
}
if(shmdt(shared_memory) == -1)
{
fprintf(stderr,"shmdt failed\n");
exit(EXIT_FAILURE);
}
if(shmctl(shmid, IPC_RMID, 0) == -1)
{
fprintf(stderr,"shmctl(IPC_RMID) Failed\n");
exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}
```

II:

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

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/shm.h>
#include "shm_com.h"
int main()
{
int running = 1;
int i,j;
void *shared_memory = (void *)0;
struct shared_use_st *shared_stuff;
int shmid;
srand((unsigned int)getpid());
shmid = shmget((key_t)1234, sizeof(struct shared_use_st), 0666 |
IPC_CREAT);
if(shmid == -1)
{
fprintf(stderr,"shmget failed\n");
exit(EXIT_FAILURE);
}
shared_memory = shmat(shmid,(void *)0,0);
if(shared_memory == (void *)-1)
{
fprintf(stderr,"shmat failed\n");
exit(EXIT_FAILURE);
}
printf("Memory attached at %X\n",(int)shared_memory);
shared_stuff = (struct shared_use_st *)shared_memory;
while(running)
{
while(shared_stuff -> written_by_you == 1)
{
sleep(1);
```

```
}
printf("Enter two numbers: \n");
scanf("%d%d",&i,&j);
shared_stuff -> aa = i;
shared_stuff -> bb = j;
shared_stuff -> written_by_you = 1;
if(shared_stuff -> aa == 0 && shared_stuff -> bb == 0)
{
running = 0;
}
}
if(shmdt(shared_memory) == -1)
{
fprintf(stderr,"shmdt failed\n");
exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}
```

SOLUTION 20:

I:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/shm.h>
#include "shm_com.h"
int main()
{
int running = 1;
void *shared_memory = (void *)0;
char buffer[BUFSIZ];
struct shared_use_st *shared_stuff;
int shmid;
```

```
srand((unsigned int)getpid());
shmid = shmget((key_t)1234, sizeof(struct shared_use_st), 0666 |
IPC_CREAT);
if(shmid == -1)
{
fprintf(stderr,"shmget failed\n");
exit(EXIT_FAILURE);
}
shared_memory = shmat(shmid,(void *)0,0);
if(shared_memory == (void *)-1)
{
fprintf(stderr,"shmat failed\n");
exit(EXIT_FAILURE);
}
printf("Memory attached at %X\n",(int)shared_memory);
shared_stuff = (struct shared_use_st *)shared_memory;
shared_stuff -> written_by_you = 0;
while(running)
{
printf("You: ");
fgets(buffer,BUFSIZ,stdin);
strncpy(shared_stuff -> some_text, buffer, TEXT_SZ);
shared_stuff -> written_by_you = 1;
while(shared_stuff -> written_by_you == 1){}
printf("Friend: %s",shared_stuff->some_text);
}
if(shmdt(shared_memory) == -1)
{
fprintf(stderr,"shmdt failed\n");
exit(EXIT_FAILURE);
}
if(shmctl(shmid, IPC_RMID, 0) == -1)
{
```

```
fprintf(stderr,"shmctl(IPC_RMID) Failed\n");
exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}
```

II:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/shm.h>
#include "shm_com.h"
int main()
{
int running = 1;
void *shared_memory = (void *)0;
struct shared_use_st *shared_stuff;
char buffer[BUFSIZ];
int shmid;
srand((unsigned int)getpid());
shmid = shmget((key_t)1234, sizeof(struct shared_use_st), 0666 |
IPC_CREAT);
if(shmid == -1)
{
fprintf(stderr,"shmget failed\n");
exit(EXIT_FAILURE);
}
shared_memory = shmat(shmid,(void *)0,0);
if(shared_memory == (void *)-1)
{
fprintf(stderr,"shmat failed\n");
exit(EXIT_FAILURE);
}
```

```
printf("Memory attached at %X\n",(int)shared_memory);
shared_stuff = (struct shared_use_st *)shared_memory;
while(running)
{
while(shared_stuff -> written_by_you==0){}
printf("Friend: %s",shared_stuff->some_text);
printf("You: ");
fgets(buffer,BUFSIZ,stdin);
strncpy(shared_stuff -> some_text, buffer, TEXT_SZ);
shared_stuff -> written_by_you = 0;
}
if(shmdt(shared_memory) == -1)
{
fprintf(stderr,"shmdt failed\n");
exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}
```

SOLUTION 21:

I:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/shm.h>
#include "shm_com.h"
int main()
{
int running = 1;
int n,i,ar[50];
void *shared_memory = (void *)0;
struct shared_use_st *shared_stuff;
int shmid;
```

```
srand((unsigned int)getpid());
shmid = shmget((key_t)1234, sizeof(struct shared_use_st), 0666 |
IPC_CREAT);
if(shmid == -1)
{
fprintf(stderr,"shmget failed\n");
exit(EXIT_FAILURE);
}
shared_memory = shmat(shmid,(void *)0,0);
if(shared_memory == (void *)-1)
{
fprintf(stderr,"shmat failed\n");
exit(EXIT_FAILURE);
}
printf("Memory attached at %X\n",(int)shared_memory);
shared_stuff = (struct shared_use_st *)shared_memory;
shared_stuff -> written_by_you = 0;
while(running)
{
printf("Enter the Number of Elements: \n");
scanf("%d",&n);
shared_stuff -> nn = n;
printf("Enter the Elements: \n");
for(i=0;i<n;i++)
{
scanf("%d",&ar[i]);
shared_stuff -> array[i] = ar[i];
}
shared_stuff -> written_by_you = 1;
while(shared_stuff -> written_by_you != 3){}
break;
}
if(shmdt(shared_memory) == -1)
```

```
{
fprintf(stderr,"shmdt failed\n");
exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}
```

II:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/shm.h>
#include "shm_com.h"
int main()
{
int running = 1;
int n,i,j,temp;
void *shared_memory = (void *)0;
struct shared_use_st *shared_stuff;
int shmid;
srand((unsigned int)getpid());
shmid = shmget((key_t)1234, sizeof(struct shared_use_st), 0666 |
IPC_CREAT);
if(shmid == -1)
{
fprintf(stderr,"shmget failed\n");
exit(EXIT_FAILURE);
}
shared_memory = shmat(shmid,(void *)0,0);
if(shared_memory == (void *)-1)
{
fprintf(stderr,"shmat failed\n");
exit(EXIT_FAILURE);
```

```
}
printf("Memory attached at %X\n",(int)shared_memory);
shared_stuff = (struct shared_use_st *)shared_memory;
while(running)
{
while(shared_stuff -> written_by_you != 1){}
n = shared_stuff -> nn;
printf("Initial Array:\n");
for(i=0;i<n;i++)
{
printf("%d ",shared_stuff -> array[i]);
}
for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
if((shared_stuff -> array[i]) > (shared_stuff -> array[j]))
{
temp = shared_stuff -> array[i];
shared_stuff -> array[i] = shared_stuff -> array[j];
shared_stuff -> array[j] = temp;
}
}
}
printf("\n\nSorted Array:\n");
for(i=0;i<n;i++)
{
printf("%d ",shared_stuff -> array[i]);
}
printf("\n\n");
shared_stuff -> written_by_you = 2;
break;
}
```

```
if(shmdt(shared_memory) == -1)
{
fprintf(stderr,"shmdt failed\n");
exit(EXIT_FAILURE);
}
if(shmctl(shmid, IPC_RMID, 0) == -1)
{
fprintf(stderr,"shmctl(IPC_RMID) Failed\n");
exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}
```

III:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/shm.h>
#include "shm_com.h"
int bs(int arr[],int n,int x)
{
int l=0,r=n,m;
while(l<=r)
{
m=(l+r)/2;
if(arr[m]==x)
return 0;
else if(x< arr[m])
r=m-1;
else
l=m+1;
}
return 1;
```

```
}
int main()
{
int running = 1;
int n,m,i,j,temp,find;
void *shared_memory = (void *)0;
struct shared_use_st *shared_stuff;
int shmid;
srand((unsigned int)getpid());
shmid = shmget((key_t)1234, sizeof(struct shared_use_st), 0666 |
IPC_CREAT);
if(shmid == -1)
{
fprintf(stderr,"shmget failed\n");
exit(EXIT_FAILURE);
}
shared_memory = shmat(shmid,(void *)0,0);
if(shared_memory == (void *)-1)
{
fprintf(stderr,"shmat failed\n");
exit(EXIT_FAILURE);
}
printf("Memory attached at %X\n",(int)shared_memory);
shared_stuff = (struct shared_use_st *)shared_memory;
while(running)
{
while(shared_stuff -> written_by_you != 2){}
n = shared_stuff -> nn;
printf("Given array:\n");
for(i=0;i<n;i++)
printf("%d ",shared_stuff -> array[i]);
while(1)
{
```

```
printf("\nEnter the Element to be searched: ");
scanf("%d",&m);
if(m==-1)
break;
find = bs(shared_stuff -> array,n-1,m);
if(find == 0)
printf("Element %d is in the array",m);
else
printf("Element %d is NOT in the array",m);
}
shared_stuff -> written_by_you = 3;
break;
}
if(shmdt(shared_memory) == -1)
{
fprintf(stderr,"shmdt failed\n");
exit(EXIT_FAILURE);
}
if(shmctl(shmid, IPC_RMID, 0) == -1)
{
fprintf(stderr,"shmctl(IPC_RMID) Failed\n");
exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}
```

SOLUTION 22.

I:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/shm.h>
#include "shm_com.h"
```

```
int main()
{
int running = 1;
int i,j,m;
void *shared_memory = (void *)0;
struct shared_use_st *shared_stuff;
int shmid;
srand((unsigned int)getpid());
shmid = shmget((key_t)1234, sizeof(struct shared_use_st), 0666 |
IPC_CREAT);
if(shmid == -1)
{
fprintf(stderr,"shmget failed\n");
exit(EXIT_FAILURE);
}
shared_memory = shmat(shmid,(void *)0,0);
if(shared_memory == (void *)-1)
{
fprintf(stderr,"shmat failed\n");
exit(EXIT_FAILURE);
}
printf("Memory attached at %X\n",(int)shared_memory);
shared_stuff = (struct shared_use_st *)shared_memory;
shared_stuff -> written_by_you = 0;
while(running)
{
printf("\nEnter the order of matrix: ");
scanf("%d",&m);
shared_stuff -> n = m;
printf("\nEnter the elements in the matrix: \n");
for(i=0;i<m;i++)
{
for(j=0;j<m;j++)
```

```
{
scanf("%d",&(shared_stuff -> arr1[i][j]));
}
}
shared_stuff -> written_by_you = 1;
while(shared_stuff -> written_by_you != 2){}
printf("\nMatrix from Producer 2:\n");
for(i=0;i<m;i++)
{
for(j=0;j<m;j++)
{
printf("%d ",shared_stuff -> arr2[i][j]);
}
printf("\n");
}
while(shared_stuff -> written_by_you != 3){}
printf("\nAdded Matrix:\n");
for(i=0;i<m;i++)
{
for(j=0;j<m;j++)
{
printf("%d\t",shared_stuff -> add[i][j]);
}
printf("\n");
}
printf("\nMultiplied Matrix:\n");
for(i=0;i<m;i++)
{
for(j=0;j<m;j++)
{
printf("%d\t",shared_stuff -> mul[i][j]);
}
printf("\n");
```

```
}
}
if(shmdt(shared_memory) == -1)
{
fprintf(stderr,"shmdt failed\n");
exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}
```

II:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/shm.h>
#include "shm_com.h"
int main()
{
int running = 1;
int i,j,m;
void *shared_memory = (void *)0;
struct shared_use_st *shared_stuff;
int shmid;
srand((unsigned int)getpid());
shmid = shmget((key_t)1234, sizeof(struct shared_use_st), 0666 |
IPC_CREAT);
if(shmid == -1)
{
fprintf(stderr,"shmget failed\n");
exit(EXIT_FAILURE);
}
shared_memory = shmat(shmid,(void *)0,0);
if(shared_memory == (void *)-1)
```

```
{
fprintf(stderr,"shmat failed\n");
exit(EXIT_FAILURE);
}
printf("Memory attached at %X\n",(int)shared_memory);
shared_stuff = (struct shared_use_st *)shared_memory;
shared_stuff -> written_by_you = 0;
while(running)
{
while(shared_stuff -> written_by_you != 1){}
m = shared_stuff -> n;
printf("\nMatrix from Producer 1:\n");
for(i=0;i<m;i++)
{
for(j=0;j<m;j++)
{
printf("%d ",shared_stuff -> arr1[i][j]);
}
printf("\n");
}
printf("\nEnter the elements in the matrix: (Order = %d)\n",m);
for(i=0;i<m;i++)
{
for(j=0;j<m;j++)
{
scanf("%d",&(shared_stuff -> arr2[i][j]));
}
}
shared_stuff -> written_by_you = 2;
while(shared_stuff -> written_by_you != 3){}
printf("\nAdded Matrix:\n");
for(i=0;i<m;i++)
{
```

```
for(j=0;j<m;j++)
{
printf("%d\t",shared_stuff -> add[i][j]);
}
printf("\n");
}
printf("\nMultiplied Matrix:\n");
for(i=0;i<m;i++)
{
for(j=0;j<m;j++)
{
printf("%d\t",shared_stuff -> mul[i][j]);
}
printf("\n");
}
}
if(shmdt(shared_memory) == -1)
{
fprintf(stderr,"shmdt failed\n");
exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}
```

III:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/shm.h>
#include "shm_com.h"
int main()
{
int running = 1 ,m,sum;
```

```
int i,j,k;
void *shared_memory = (void *)0;
struct shared_use_st *shared_stuff;
int shmid;
srand((unsigned int)getpid());
shmid = shmget((key_t)1234, sizeof(struct shared_use_st), 0666 |
IPC_CREAT);
if(shmid == -1)
{
fprintf(stderr,"shmget failed\n");
exit(EXIT_FAILURE);
}
shared_memory = shmat(shmid,(void *)0,0);
if(shared_memory == (void *)-1)
{
fprintf(stderr,"shmat failed\n");
exit(EXIT_FAILURE);
}
printf("Memory attached at %X\n",(int)shared_memory);
shared_stuff = (struct shared_use_st *)shared_memory;
while(running)
{
while(shared_stuff -> written_by_you != 1){}
m = shared_stuff -> n;
printf("\nMatrix from Producer 1:\n");
for(i=0;i<m;i++)
{
for(j=0;j<m;j++)
{
printf("%d ",shared_stuff -> arr1[i][j]);
}
printf("\n");
}
```

```
while(shared_stuff -> written_by_you != 2){}
printf("\nMatrix from Producer 2:\n");
for(i=0;i<m;i++)
{
for(j=0;j<m;j++)
{
printf("%d ",shared_stuff -> arr2[i][j]);
}
printf("\n");
}
//Matrix addition and multiplication:
for(i=0;i<m;i++)
{
for(j=0;j<m;j++)
{
shared_stuff -> add[i][j]= shared_stuff -> arr1[i][j] + shared_stuff -> arr2[i][j];
sum=0;
for(k=0;k<m;k++)
{
sum+= (shared_stuff -> arr1[i][k])*(shared_stuff -> arr2[k][j]);
}
shared_stuff -> mul[i][j] = sum;
}
printf("\n");
}
shared_stuff -> written_by_you = 3;
break;
}
if(shmdt(shared_memory) == -1)
{
fprintf(stderr,"shmdt failed\n");
exit(EXIT_FAILURE);
}
```

```
if(shmctl(shmid, IPC_RMID, 0) == -1)
{
fprintf(stderr,"shmctl(IPC_RMID) Failed\n");
exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}
```

SOLUTION 23:

I:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/shm.h>
#include "shm_com.h"
int main()
{
int running = 1,i;
char c;
FILE *fp;
char word[10];
void *shared_memory = (void *)0;
struct shared_use_st *shared_stuff;
int shmid;
srand((unsigned int)getpid());
shmid = shmget((key_t)1234, sizeof(struct shared_use_st), 0666 |
IPC_CREAT);
if(shmid == -1)
{
fprintf(stderr,"shmget failed\n");
exit(EXIT_FAILURE);
}
shared_memory = shmat(shmid,(void *)0,0);
```

```
if(shared_memory == (void *)-1)
{
fprintf(stderr,"shmat failed\n");
exit(EXIT_FAILURE);
}
printf("Memory attached at %X\n",(int)shared_memory);
shared_stuff = (struct shared_use_st *)shared_memory;
shared_stuff -> written_by_you = 0;
while(running)
{
if(shared_stuff -> written_by_you == 2)
{
fp = fopen("1.txt","r");
if(fp == NULL)
{
printf("Cannot open the file\n");
}
else
{
while(fscanf(fp,"%s",word) != EOF)
{
printf("%s\n",word);
}
}
fclose(fp);
fp = fopen("2.txt","r");
if(fp == NULL)
{
printf("Cannot open the file\n");
}
else
{
while(fscanf(fp,"%s",word) != EOF)
```

```
{
printf("%s\n",word);
}
}
fclose(fp);
fp = fopen("3.txt","r");
if(fp == NULL)
{
printf("Cannot open the file\n");
}
else
{
while(fscanf(fp,"%s",word) != EOF)
{
printf("%s\n",word);
}
}
fclose(fp);
fp = fopen("4.txt","r");
if(fp == NULL)
{
printf("Cannot open the file\n");
}
else
{
while(fscanf(fp,"%s",word) != EOF)
{
printf("%s\n",word);
}
}
fclose(fp);
fp = fopen("5.txt","r");
if(fp == NULL)
```

```
{
printf("Cannot open the file\n");
}
else
{
while(fscanf(fp,"%s",word) != EOF)
{
printf("%s\n",word);
}
}
fclose(fp);
sleep(1);
shared_stuff -> written_by_you = 0;
}
if(shared_stuff -> written_by_you == 0)
{
c = 0;
printf("Enter the text :\n");
while(c != '\n')
{
scanf("%s",(shared_stuff -> text)[i]);
c = getchar();
i++;
}
shared_stuff -> flag = i;
shared_stuff -> written_by_you = 1;
}
}
if(shmdt(shared_memory) == -1)
{
fprintf(stderr,"shmdt failed\n");
exit(EXIT_FAILURE);
}
```

```
if(shmctl(shmid, IPC_RMID, 0) == -1)
{
fprintf(stderr,"shmctl(IPC_RMID) Failed\n");
exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}
```

II:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/shm.h>
#include "shm_com.h"
int main()
{
int running = 1,i,l;
char *word;
FILE *fp;
word = (char*)malloc(sizeof(char)*10);
void *shared_memory = (void *)0;
struct shared_use_st *shared_stuff;
int shmid;
srand((unsigned int)getpid());
shmid = shmget((key_t)1234, sizeof(struct shared_use_st), 0666 |
IPC_CREAT);
if(shmid == -1)
{
fprintf(stderr,"shmget failed\n");
exit(EXIT_FAILURE);
}
shared_memory = shmat(shmid,(void *)0,0);
if(shared_memory == (void *)-1)
```

```
{
fprintf(stderr,"shmat failed\n");
exit(EXIT_FAILURE);
}
printf("Memory attached at %X\n",(int)shared_memory);
shared_stuff = (struct shared_use_st *)shared_memory;
while(running)
{
if(shared_stuff -> written_by_you == 1)
{
printf("Reading each word.....\n");
for(i = 0; i < shared_stuff -> flag; i++)
{
printf("%s\n",(shared_stuff->text)[i]);
l = strlen((shared_stuff -> text)[i]);
switch(l)
{
case 1: fp = fopen("1.txt","a");
if(fp == NULL)
{
printf("Cannot open the file\n");
}
fprintf(fp,"%s",(shared_stuff -> text)[i]);
fputc('\n',fp);
fclose(fp);
break;
case 2: fp = fopen("2.txt","a");
if(fp == NULL)
{
printf("Cannot open the file\n");
}
fprintf(fp,"%s",(shared_stuff -> text)[i]);
fputc('\n',fp);
```

```
fclose(fp);
break;
case 3: fp = fopen("3.txt","a");
if(fp == NULL)
{
printf("Cannot open the file\n");
}
fprintf(fp,"%s",(shared_stuff -> text)[i]);
fputc('\n',fp);
fclose(fp);
break;
case 4: fp = fopen("4.txt","a");
if(fp == NULL)
{
printf("Cannot open the file\n");
}
fprintf(fp,"%s",(shared_stuff -> text)[i]);
fputc('\n',fp);
fclose(fp);
break;
case 5: fp = fopen("5.txt","a");
if(fp == NULL)
{
printf("Cannot open the file\n");
}
fprintf(fp,"%s",(shared_stuff -> text)[i]);
fputc('\n',fp);
fclose(fp);
break;
default : break;
}
}
shared_stuff->written_by_you = 2;
```

```
}
}
if(shmdt(shared_memory) == -1)
{
fprintf(stderr,"shmdt failed\n");
exit(EXIT_FAILURE);
}
if(shmctl(shmid, IPC_RMID, 0) == -1)
{
fprintf(stderr,"shmctl(IPC_RMID) Failed\n");
exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}
```