1)odd/even

```
echo "Enter a number : "
read n
rem=$(( $n % 2 ))
if [ $rem -eq 0 ]
then
echo "$n is even number"
else
echo "$n is odd number"
fi
```

2)factorial

```
echo "Enter a number"
read num
fact=1
while [ $num -gt 1 ]
do
fact=$((fact * num))
num=$((num - 1))
done
echo Factorial=$fact
```

3)calculator

```
sum=0
i="y"
echo "Enter first number :"
read n1
echo "Enter second number :"
read n2
while [ $i = "y" ]
do
echo "1.Addition"
echo "2.Subtraction"
echo "3.Multiplication"
echo "4.Division"
echo "Enter your choice"
read ch
case $ch in
1)sum=`expr $n1 + $n2`
echo "Sum ="$sum;;
2)sub=`expr $n1 - $n2`
echo "Sub = "$sub;;
3)mul=`expr $n1 \* $n2`
echo "Mul = "$mul;;
4)div=`echo $n1 / $n2 | bc -l`
echo "Div = "$div;;
*)echo "Invalid choice";;
esac
echo "Do u want to continue ?"
read i
if [ $i != "y" ]
then
exit
fi
done
```

4) **Inter-process communication between related process using pipes**

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

int main() {
int pipefds[2];
int returnstatus;
int pid;
char writemessages[2][20]={"Hi", "Hello"};
char readmessage[20];
returnstatus = pipe(pipefds);
if (returnstatus == -1) {
printf("Unable to create pipe\n");
return 1;
}
pid = fork();

// Child process
if (pid == 0) {
read(pipefds[0], readmessage, sizeof(readmessage));
printf("Child Process - Reading from pipe – Message 1 is %s\n", readmessage);
read(pipefds[0], readmessage, sizeof(readmessage));
printf("Child Process - Reading from pipe – Message 2 is %s\n", readmessage);
} else { //Parent process
printf("Parent Process - Writing to pipe - Message 1 is %s\n", writemessages[0]);
write(pipefds[1], writemessages[0], sizeof(writemessages[0]));
printf("Parent Process - Writing to pipe - Message 2 is %s\n", writemessages[1]);
write(pipefds[1], writemessages[1], sizeof(writemessages[1]));
}
return 0;
}
```

5) **FCFS Scheduling Program in C**

```
#include <stdio.h>
void main()
{
int pid[15];
int bt[15];
int n,i;
printf("Enter the number of processes: ");
scanf("%d",&n);

printf("Enter process id of all the processes: ");
for(i=0;i<n;i++)
{
scanf("%d",&pid[i]);
}

printf("Enter burst time of all the processes: ");
```

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

        int wt[n];
        wt[0]=0;

        //for calculating waiting time of each process
        for(i=1; i<n; i++)
        {
        wt[i]= bt[i-1]+ wt[i-1];
        }
printf("Process ID    Burst Time    Waiting Time    TurnAround Time\n");
        float twt=0.0;
        float tat= 0.0;
        for(i=0; i<n; i++)
        {
        printf("%d\t\t", pid[i]);
        printf("%d\t\t", bt[i]);
        printf("%d\t\t", wt[i]);

        //calculating and printing turnaround time of each process
        printf("%d\t\t", bt[i]+wt[i]);
        printf("\n");

        //for calculating total waiting time
        twt += wt[i];

        //for calculating total turnaround time
        tat += (wt[i]+bt[i]);
        }
        float att,awt;

        //for calculating average waiting time
        awt = twt/n;

        //for calculating average turnaround time
        att = tat/n;
        printf("Avg. waiting time= %f\n",awt);
        printf("Avg. turnaround time= %f",att);
        }
```

6) **C program to implement priority scheduling(preemptive)**

```
#include <stdio.h>

void swap(int *a,int *b)
{
int temp=*a;
*a=*b;
*b=temp;
}
int main()
{
int n;
printf("Enter Number of Processes: ");
scanf("%d",&n);
```

```
int burst[n],priority[n],index[n];
for(int i=0;i<n;i++)
{
printf("Enter Burst Time and Priority Value for Process %d: ",i+1);
scanf("%d %d",&burst[i],&priority[i]);
index[i]=i+1;
}
for(int i=0;i<n;i++)
{
int temp=priority[i],m=i;

for(int j=i;j<n;j++)
{
if(priority[j] > temp)
{
temp=priority[j];
m=j;
}
}

swap(&priority[i], &priority[m]);
swap(&burst[i], &burst[m]);
swap(&index[i],&index[m]);
}

int t=0;

printf("Order of process Execution is\n");
for(int i=0;i<n;i++)
{
printf("P%d is executed from %d to %d\n",index[i],t,t+burst[i]);
t+=burst[i];
}
printf("\n");
printf("Process Id\tBurst Time\tWait Time\n");
int wait_time=0;
int total_wait_time = 0;
for(int i=0;i<n;i++)
{
printf("P%d\t\t%d\t\t%d\n",index[i],burst[i],wait_time);
total_wait_time += wait_time;
wait_time += burst[i];
}

float avg_wait_time = (float) total_wait_time / n;
printf("Average waiting time is %f\n", avg_wait_time);

int total_Turn_Around = 0;
for(int i=0; i < n; i++){
total_Turn_Around += burst[i];
}
float avg_Turn_Around = (float) total_Turn_Around / n;
printf("Average TurnAround Time is %f",avg_Turn_Around);
return 0;
}
```

7) First Fit

```
#include<stdio.h>
void main()
{
int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;
for(i = 0; i < 10; i++)
{
flags[i] = 0;
allocation[i] = -1;
}
printf("Enter no. of blocks: ");
scanf("%d", &bno);
printf("\nEnter size of each block: ");
for(i = 0; i < bno; i++)
scanf("%d", &bsize[i]);
printf("\nEnter no. of processes: ");
scanf("%d", &pno);
printf("\nEnter size of each process: ");
for(i = 0; i < pno; i++)
scanf("%d", &psize[i]);
for(i = 0; i < pno; i++)        //allocation as per first fit
for(j = 0; j < bno; j++)
if(flags[j] == 0 && bsize[j] >= psize[i])
{
allocation[j] = i;
flags[j] = 1;
break;
}
//display allocation details
printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");
for(i = 0; i < bno; i++)
{
printf("\n%d\t\t%d\t\t", i+1, bsize[i]);
if(flags[i] == 1)
printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);
else
printf("Not allocated");
}
}
```

8) Best Fit

```
#include<stdio.h>
void main()
{
int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;
static int barray[20],parray[20];
printf("\n\t\t\tMemory Management Scheme - Best Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of processes:");
scanf("%d",&np);
printf("\nEnter the size of the blocks:-\n");
```

```
for(i=1;i<=nb;i++)
{
printf("Block no.%d:",i);
scanf("%d",&b[i]);
}
printf("\nEnter the size of the processes :-\n");
for(i=1;i<=np;i++)
{
printf("Process no.%d:",i);
scanf("%d",&p[i]);
}
for(i=1;i<=np;i++)
{
for(j=1;j<=nb;j++)
{
if(barray[j]!=1)
{
temp=b[j]-p[i];
if(temp>=0)
if(lowest>temp)
{
parray[i]=j;
lowest=temp;
}
}
}
fragment[i]=lowest;
barray[parray[i]]=1;
lowest=10000;
}
printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment");
for(i=1;i<=np && parray[i]!=0;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i]);
}
```

9) C program to illustrate close system Call

```
#include<stdio.h>

#include<fcntl.h>

int main()

{

// assume that foo.txt is already created

int fd1 = open("foo.txt", O_RDONLY, 0);

close(fd1);



// assume that baz.tzt is already created

int fd2 = open("baz.txt", O_RDONLY, 0);
```

```
printf("fd2 = % d\n", fd2);
exit(0);
}
```

10) C program to illustrate open system call

```
#include<stdio.h>
#include<fcntl.h>
#include<errno.h>
extern int errno;
int main()
{
// if file does not have in directory
// then file foo.txt is created.
int fd = open("foo.txt", O_RDONLY | O_CREAT);

printf("fd = %d\n", fd);

if (fd ==-1)
{
// print which type of error have in a code
printf("Error Number % d\n", errno);

// print program detail "Success or failure"
perror("Program");
}
return 0;
}
```

11) Sum of digits

```
echo "Enter a number"
read num

sum=0

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

echo $sum
```