Find the greatest of given three numbers

```
echo "Enter the 3 elements one byone:"
read a
read b
read c
if [ $a -gt $b -a $a -gt $c ]
then
echo "$a is greater"
elif [ $b -gt $c ]
then
echo "$b is greater"
else
echo "$c is greater”
fi
```

Output:

```
Enter the 3 elements one by one:
10
20
100
100 is greater
```

Find out the max and min number of a given series

```
echo "Enter the no of elements in the array:"
read n
echo "Enter the elements one by one:"
```

```
i=0
while [ $i -lt $n ]
do
read array[$i]
let i++
done
len=${#array[*]}
echo "The array has $len members They are"
i=0
lenwhile [ $i -lt $len ]
do
echo "$i:${array[$i]}"
let i++
done
i=0
j=1
min=${array[$i]}
max=${array[$i]}
while [ $j -lt $len ]
do
sec=${array[$j]}
if [ $min -gt $sec ]
then
min=$sec
elif [ $max -lt $sec ]
```

```
then
max=$sec
fi
let j++
done
echo "Minimum=$min"
echo "Maximum=$max"
```

Output:

```
Enter the no of elements in the array:
6
Enter the elements one by one:
4
78
3
56
789
45
The array has 6 members they are
0:4
1:78
2:3
3:56
4:789
5:45
Minimum=3
```

Maximum=789

Palindrome Checking

```
echo "Enter the string:"
read str
len=${#str}
k=$len-1
i=0
flag=1
while [ $i -le $((len / 2))
do
s1=${str:$i:1}
s2=${str:$k-i:1}
if [ $s1 != $s2 ]
then
flag=0
fi
let i++
done
if [ $flag -eq 1 ]
then
echo "Palindrome"
else
echo "Not Palindrome"
```

```
fi
```

Find the LENGTH of given string

```
echo "Enterthe string "
read str
len=${#str}
echo "$len”
```

Develop a Calculator Application

```
j=1
while [ $j -eq 1 ]
do
echo "Enter the First Operand;"
read f1
echo "Enter the secondoperand:"
read f2
echo "1-> Addition"
echo "2-> Subtraction"
echo "3-> Multiplication"
echo "4-> Division"
echo "Enter your choice"
read n
case "$n" in
1)
echo "Addition"
```

```
f3=$((f1+f2))
echo "The result is:$f3"
;;
2)
echo "Subtraction"
let "f4=$f1 - $f2"
echo "The result is:$f4"
;;
3)
echo "Multiplication"
let "f5=$f1 * $f2"
echo "The result is:$f5"
;;
4)
echo "Division"
let "f6=$f1 / $f2"
echo "The result is:$f6"
;;
esac
echo "Do you want to continue(press:1 otherwise press any key to
quit)"
read j
done
```

Implement Deadlock Avoidance Algorithm using C Language

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

```
#include<stdlib.h>

main()

{

const int max = 10; //max multiplicity

int i, j, z; //counters

int P, L, N; //P=processes, L= FREE, N= maximum multiplity

int A[10], R[10], M[10]; //A= allocates, R=request, M= maximum

requests

int RS[10]; //RS= folowing request

int LS[10]; //free resurces after the i-allocation

int CP[10]={0}; //CP= processes classification

int MAX[10];

int lim; //max limit

do

{

printf("\nHOW MANY PROCESSES DO YOU WANT? ");

scanf("%d",&P);

} while(P>max);

printf("\nENTER THE TOTAL NO OF RESOURCES: ");

scanf("%d",&N);

for(i=1;i<=P;i++)

{

}

L=N;

printf("\nHOW MANY UNIT HAS THE P%d PROCESS?",i);
```

```
scanf("%d",&A[i]);
printf("\nHOW MANY UNIT REQUEST THE P%d
PROCESS?",i);
scanf("%d",&R[i]);
printf("\nWHAT IS THE MAX NUMBER OF UNITS OF THE
P%d PROCESS?",i);
scanf("%d",&M[i]);
printf("\n\n");
for(i=1; i<=P; i++)
L-=A[i]; //nuber of units stillidle
for(i=1; i<=P; i++)
{
if(R[i]<=L) //is secure?
{
RS[i]=M[i]-A[i]-R[i]; //next request
LS[i]=L-R[i]; //idle resuorces after the allocation
}
else //isn't secure
{
LS[i]=-1;
}
}
for(i=1; i<=P; i++)
MAX[i]=-1; //or 0???
z=1;
```

```
for(i=1; i<=P; i++)
{
if(i==1)
{
for(j=1; j<=P; j++)
{
if(LS[j]>=MAX[z])
{
MAX[z]=LS[j];
CP[z]=j;
lim=MAX[z];
}}}
Else
{
for(j=1; j<=P; j++)
{
if((LS[j]>=MAX[z])&&(LS[j]<lim))
{
MAX[z]=LS[j];
CP[z]=j;
lim=MAX[z];
}
}
}
Z++;
```

```
}
for(i=1; i<=P; i++)
{
if(LS[i]!=-1)
printf("\nTHE PROCESS %d IS THE %dstCHOICE\n",i,CP[i]);
else
printf("THE PROCESS %d IS NOT SECURE",i);
}
}
```

Implement Multithreading Concepts

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 3
void *BusyWork(void *null)
{
int i;
double result=0.0;
for (i=0; i<1000000; i++)
{
result = result + (double)random();
}
printf("result = %e\n",result);
pthread_exit ((void *) 0);
}
int main (int argc, char *argv[])
```

```
{
pthread_t thread[NUM_THREADS];
pthread_attr_t attr;
int rc, t;
void *status;
/* Initialize and set thread detached attribute */
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_JOINABLE);
for(t=0; t<NUM_THREADS; t++)
{
printf("Creating thread %d\n", t);
rc = pthread_create(&thread[t], &attr, BusyWork,
NULL);
if (rc)
{
printf("ERROR; return code from
thread_create()is %d\n", rc);
exit(-1);
}
}
/* Free attribute and wait for the other threads */
pthread_attr_destroy(&attr);
for(t=0; t<NUM_THREADS; t++)
{
rc = pthread_join(thread[t], &status);
```

```
if (rc)
{
printf("ERROR; return code from
pthread_join()is %d\n", rc);
exit(-1);
}
printf("Completed join with thread %d status=
%ld\n",t,(long)status);
}
pthread_exit(NULL);
}
```

First Come First Served (FCFS) Scheduling

```
#include<stdio.h>
#include<string.h>
int main()
{
Int n,i,j,temp,exe[15],arrival[15],wait[15],tat[15],
timearray[15];
float twt=0,ttat=0,avgwt=0,avgtat=0;
char pid[15][15],tname[15],namearray[15][15];
printf("\n Enter the number of process:");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
printf("\nEnter the processname:");
```

```
scanf("%s",&pid[i]);
printf("\nEnter the executiontime:");
scanf("%d",&exe[i]);
printf("\n Enter the arrival time:");
scanf("%d",&arrival[i]);
}
for(i=1;i<=n;i++)
{
for(j=i;j<=n;j++)
{
if(arrival[i]>arrival[j])
{
strcpy(tname,pid[i]);
strcpy(pid[i],pid[j]);
strcpy(pid[j],tname);
temp=exe[i];
exe[i]=exe[j];
exe[j]=temp;
temp=arrival[i];
arrival[i]=arrival[j];
arrival[j]=temp;
}
}
}
timearray[0]=0;
```

```
for(i=1,j=1;i<=n;i++,j++)
{
if(arrival[i]>timearray[j-1])
{
timearray[j]=arrival[i];
strcpy(namearray[j],"idle");
i=i-1;
}
else
{
timearray[j]=timearray[j-1]+exe[i];
strcpy(namearray[j],pid[i]);
}
}
for(i=1,j=1;i<=n;i++,j++)
{
if(strcmp(namearray[j],"idle")==0)
{
i=i-1;
}
else
{
wait[i]=timearray[j-1]-arrival[i];
tat[i]=timearray[j]-arrival[i];
twt=twt+
```

```
wait[i];
ttat=ttat+tat[i];
}
}
avgwt=twt/n;
avgtat=ttat/n;
printf("\n Name arrival Execution Waiting
Turnaround Time\n");
for(i=1,j=1;i<=n;i++,j++)
{
if(strcmp(namearray[j],"idle")==0)
{
i=i-1;
}
else
{
printf("\n%s",namearray[j]);
printf("\t%d",arrival[i]);
printf(" %d to %d",timearray[j-
1],timearray[j]);
printf(" %d",wait[i]);
printf(" %d",tat[i]);
printf("\n");
}
}
```

```
printf("\n Average waiting time:%f",avgwt);

printf("\n Average turnaround time:%f\n",avgtat);

return(0);

}
```