```
do
{
        if(pt[ i ]>timer)
        {
                rt=pt[ i ] – timer;
                strcpy(p[n],p[ i ]);
                pt[n]=rt;
                et[ i ]=timer;
                n++;
        }
        else
        {
                et[ i ]=pt[i];
        }
        i++;
        wt[i]=wt[i-1]+et[i - 1];
}while(i<n);

count=0;

for(i=0;i<m;i++)
{
        for(j=i+1;j<=n;j++)
        {
                if(strcmp(p[i],p[j])==0)
                {
                        count++;
                        found=j;
                }
```

```
        }
        if(found!=0)
        {
                wt[i]=wt[found]-(count*timer);
                count=0;
                found=0;
        }
}
for(i=0;i< m ;i++)
{
        totwt+=wt[i];
}
avgwt=(float)totwt/m;
for(i=0;i<m;i++)
{
        totta+=wt[i]+pt[i];
}
avgta=(float)totta/m;

printf("\tProcess name\tProcess time\t\tWaiting time\n");
for(i=0;i<m;i++)
{
        printf("\t%s\t\t\t%d\t\t\t%d\n",p[i],pt[i],wt[i]);
}
printf("\nTotal waiting time = %d\n",totwt);
printf("\nTotal average waiting time = %f\n",avgwt);
printf("\nToatlturn around time = %d\n",totta);
printf("\nAvearageturn around time = %f\n",avgta);
}
```

Prog :10 **Round Robin** USN:2VD

OUTPUT

Enter the number of processes

3

Enter the quantum time

2

Enter the process name:

a

Enter the processing time

2

Enter the process name:

b

Enter the processing time

2

Enter the process name:

c

Enter the processing time

4

Process name	Process time	Waiting time
a	2	0
b	2	2
c	4	4

Total waiting time = 6

Total average waiting time = 2.000000

Totalturn around time = 14

Average turn around time = 4.666667

Prog :10 Shortest Remaining Time USN:2VD

```
#include <stdio.h>
#include<stdlib.h>
#include<string.h>
int main( )
{
        char  p[ 10 ][ 10 ] , temp[ 5 ] ;
        int   tot = 0 , wt[ 10 ] , pt[ 10 ] , i , j , n , temp1 , ta[ 10 ] , totta = 0 ;
        float   avg = 0 , avgta = 0 ;
        printf( "\nEntertrhe number of processes\n " ) ;
        scanf( "%d " , &n ) ;
        for( i = 0 ; i < n ; i++ )
        {
                printf( "\nEntre the %d process\n" , i + 1 ) ;
                scanf( " %s " , &p[ i ]) ;
                printf( "\nEnter the process time\n" ) ;
                scanf( " %d " , &pt[ i ]) ;

        }
        for( i = 0 ; i < n ; i++ )
        {
                for( j = i + 1 ; j < n ; j++ )
                {
                        if( pt[ i ] >pt[ j ] )
                        {
                                temp1 = pt[ i ] ;
                                pt[ i ] = pt[ j ] ;
                                pt[ j ] = temp1 ;
                                strcpy( temp , p[ i ] ) ;
                                strcpy( p[ i ] , p[ j ] ) ;
                                strcpy( p[ j ] , temp) ;

                        }
                }
        }
        wt[ 0 ] = 0 ;
        ta[ 0 ] = pt[ 0 ] ;
        for( i = 1 ; i < n ; i++ )
        {
                wt[ i ] = wt[ i - 1 ] + pt[ i - 1 ] ;
                tot = tot + wt[ i ] ;
                ta[ i ] = pt[ i ] + wt[ i -1 ] + pt[ i - 1 ] ;
                totta = totta + ta[ i ] ;
        }
        avg = ( float )tot / n ;
        avgta = ( float )totta / n ;
        printf( "\n\tProcess name\tProcess time\tWaiting time\tTurn around time\n ") ;
        for( i = 0 ; i < n ; i++ )
        {
```

```
        printf( "\t\t%s\t\t%d\t\t%d\t\t%d\n " , p[ i ] , pt[ i ] , wt[ i ] , ta[ i ]) ;
        }
        printf( "\nTotal waiting time  = %d\n" , tot)  ;
        printf( "\nAverage Waiting Time  = %f\n" , avg ) ;
        printf("\nTotalturn around time = %d\n" , totta) ;
        printf("\nAverageTurn around time  = %f\n " , avgta ) ;
}
```

OUTPUT

```
Enter trhe number of processes
3

Enter the process 1
a

Enter the process time
2

Enter the process 2
b

Enter the process time
2

Enter the process 3
c

Enter the process time
4

Process name    Process time    Waiting time    Turn around time
        a               2               0               2
        b               2               2               4
        c               4               4               8

Total waiting time = 6
Average Waiting Time = 2.000000
Total turn around time = 12
Average Turn around time = 4.000000
```

Design, develop and run a program to implement the Banker's Algorithm. Demonstrate its working with different data values.

```
#include <stdio.h>
#include<stdlib.h>
#define true 1
#define false 0
int  m , n , i , j , count = 0 , process ;
int  max[ 10 ][ 10 ] , alloc[ 10 ][ 10 ] ;
int  need[ 10 ][ 10 ] , c[ 10 ] , avail[ 10 ] , finish[ 10 ] ;

void readtable( int t[ 10 ][ 10 ] )
{
        for( i = 0 ; i < m ; i++ )
        {
                for( j = 0 ; j < n ; j++ )
                {
                        scanf( " %d " , &t[ i ][ j ]) ;
                }
        }
}

void printtable( int t[ 10 ][ 10 ] )
{
        for( i = 0 ; i < m ; i++ )
        {
                for( j = 0 ; j < n ; j++ )
                {
                        printf( "\t%d " , t[ i ][ j ]) ;
                }
                printf( " \n " ) ;
        }
}

void readvector( int v[ 10 ] )
{
        for( j =  0 ; j < n  ; j++ )
        {
                scanf( " %d " , &v[ j ] ) ;
        }
}

void printvector( int v[ 10 ] )
{
        for( j = 0 ; j < n ; j++ )
        {
```

```
                printf( " \t%d " , v[ j ] ) ;
        }
}

void init( )
{
        printf( "\nEnter the number of processes\n" ) ;
        scanf( " %d " , &m ) ;
        printf( "\nEnter the number of resources\n ") ;
        scanf( " %d " , &n) ;
        printf(" \nEnter the claim(MAX) table\n " ) ;
        readtable( max ) ;
        printf(" \nEnter the allocation table\n ") ;
        readtable( alloc ) ;
        printf(" \nEnter the max units of each resourses\n" ) ;
        readvector( c ) ;
        for( i = 0 ; i < n ; i++ )
        {
                finish[ i ] = false ;
        }
}

void findavail( )
{
        int sum ;
        for( j = 0 ; j < n ; j++ )
        {
                sum = 0 ;
                for( i = 0 ; i < m ; i++ )
                {
                        sum = sum + alloc[ i ][ j ] ;
                }
                avail[ j ] = c[ j ] - sum ;
        }
}

void findneed( )
{
        printf( "Need\n" ) ;
        for( i = 0 ; i < m ; i++ )
        {
                for( j = 0 ; j < n ; j++ )
                {
                        need[ i ][ j ] = max[ i ][ j ] - alloc[ i ][ j ] ;
                printf(" %d\t " , need[ i ][ j ] ) ;
                }
```

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

void selectprocess( )
{
        int  flag ;
        for( i = 0 ; i < m ; i++ )
        {
                for( j = 0 ; j < n ; j++ )
                {
                        if( need[ i ][ j ] <= avail[ j ] )
                                flag = 1 ;
                        else
                        {
                                flag = 0 ;
                                break ;
                        }
                }
                if( (flag == 1 ) && ( finish[ i ] == false ) )
                {
                        processs = i ;
                        count++ ;
                        break ;
                }
        }
        printf( "\nCurrent Status is :\n " ) ;
        printtable( alloc ) ;
        if( flag == 0 )
        {
                printf( "\nSystem is in unsafe state\n" ) ;
                exit( -1 ) ;
        }
        printf( "\nSystem is in safe state\n " ) ;
}

void executeprocess( int  p )
{
        printf(" \nExecuting process is %d\n " , p ) ;
        printtable( alloc ) ;
}

void releaseresource( )
{
        for( j = 0 ; j < n ; j++ )
                avail[ j ] = avail[ j ] + alloc[ process ][ j ] ;
```

```
        for( j = 0 ; j < n ; j++)
        {
                alloc[ process ][ j ] = 0 ;
                need[ process ][ j ] = 0 ;
        }
}

main( )
{
        int  i ;
        init( ) ;
        findavail( ) ;
        findneed( ) ;
        for( i = 0 ; i <= m ; i++)
        {
                if( count <= m )
                {
                        selectprocess( ) ;
                        finish[ process ] = true ;
                        executeprocess( process ) ;
                        releaseresource( ) ;
                }
        }
        printf( "\nAll process executed correctly\n" ) ;
}
```

OUTPUT

```
Enter the number of processes
5
Enter the number of resources
3
Enter the claim(MAX) table
7    5    3
3    2    2
9    0    2
2    2    2
4    3    3
Enter the allocation table
0    1    0
2    0    0
3    0    2
2    1    1
0    0    2
Enter the max units of each resource
10   5    7
```

```
Need
7    4    3
1    2    2
6    0    0
0    1    1
4    3    1
Current Status is:
     0    1    0
     2    0    0
     3    0    2
     2    1    1
     0    0    2
System is in safe state
Executing process is 1
     0    1    0
     2    0    0
     3    0    2
     2    1    1
     0    0    2
Current Status is:
     0    1    0
     0    0    0
     3    0    2
     2    1    1
     0    0    2
System is in safe state
Executing process is 3
     0    1    0
     0    0    0
     3    0    2
     2    1    1
     0    0    2
Current Status is :
     0    1    0
     0    0    0
     3    0    2
     0    0    0
     0    0    2
System is in safe state
Executing process is 0
     0    1    0
     0    0    0
     3    0    2
     0    0    0
     0    0    2
Current Status is :
     0    0    0
     0    0    0
     3    0    2
     0    0    0
     0    0    2
System is in safe state
```

Yasmeen. A. Mulla

Prog : ~~10~~ 7 **Round Robin** USN:2VD[illegible]

Design, develop and execute a program in C/C++ to simulate the working of Shortest Remaining time and Round Robin Scheduling algorithms. Experiment with different Quantum sizes for the Round Robin algorithm. In all cases, determine the average turn Around time. Input can be read from keyboard or from a file.

```
#include<stdio.h>
#include<string.h>
#include<math.h>
main()
{
        char p[10][5];
        intet[10],wt[10],timer,count,pt[10] ;
        intrt, i ,j,totwt=0,t,found=0,m,n,totta=0;
        float avgwt,avgta=0;
        printf("\nEnter the number of processes\n");
        scanf("%d",&n);
        printf("Enter the quantum time\n");
        scanf("%d",&timer);
        for( i =0; i <n;i++)
        {
                printf("\nEnter the process name:\n");
                scanf("%s",&p[ i ]);
                printf("Enter the processing time\n");
                scanf("%d",&pt[ i ]);
        }
        printf("\n");
        m=n;
        wt[0]=0;
        i =0;
```