Q Develop a C program to do addition subtraction & multiplication of two matrices

→
```
#include<stdio.h>
int main()
{
    int a[3][3], b[3][3], c[3][3],
        d[3][3], e[3][3];
    int i, j, m, n, p, q, k;
    printf("Enter no of rows & columns of
            matrix A");
    scanf("%d%d", &m, &n);
    printf("Enter the rows & columns of
            matrix B");
    scanf("%d%d", &p, &q);
    if (m!=p || n!=q)
{   printf("Addition & subtraction not possible");
    else if (n!=p)
    {printf("Multiplication is not possible");}
else {printf("Enter the values of matrix A");
    for(i=0; i<m; i++)
        {for(j=0; j<n; j++)
            {scanf("%d", &a[i][j]);
            }}
```

```
printf("Enter the values of matrix B");
    for (i=0; i<p; i++)
     { for (j=0; j<q; j++)
       { scanf("%d", &b[i][j]); }}

printf("Addition and Subtraction of A & B
 matrices");
for (i=0; i<m; i++)
{ for (j=0; j<n; j++)
   { c[i][j] = a[i][j] + b[i][j];
     d[i][j] = a[i][j] - b[i][j];
     printf("%d", &c[i][j]);
     printf("%d", &d[i][j]);
   }}
printf("Multiplication of A and B matrices");
 for (i=0; i<m; i++)
    { for (j=0; j<q; j++)
       { e[i][j] = 0;
      for (k=0; k<n; k++)
        { e[i][j] = e[i][j] + a[i][k]
                  * b[k][i]; }
      printf("%d", &e[i][j]);
    }
  }
}
```

OUTPUT

Enter rows & coln of matrix A : 2 2

Enter rows & coln of matrix B : 2 2

Enter the values of matrix A :

1 2

3 4

Enter the values of matrix B :

5 6

7 8

Addition & Subtraction of A and B matrices

6 8 -4 -4

10 12 -4 -4

Multiplication of A & B matrices 19 22

43 50

Q Write a c code for FCFS and SJF

```
#include<stdio.h>
int n, i, j, pos, temp, choice, total=0
int Burst_time[20], Arrival_time[20], Waiting_time[20],
Turn_around_time[20], process[20];
float avg_Turn_around_time=0, avg_Waiting_time=0;

void FCFS()
{ int total_waiting_time=0, total_turnaround_time=0;
  int current_time=0;
  for(i=0; i<n-1; i++){
    for(j=i+1; j<n; j++){
      if(Arrival_time[i] > Arrival_time[j])
      { temp = Arrival_time[i]
        Arrival_time[i] = Arrival_time[j];
        Arrival_time[j] = temp;
        temp = Burst_time[i];
        Burst_time[i] = Burst_time[j];
        Burst_time[j] = temp;
        temp = process[i];
        process[i] = process[j];
        process[j] = temp;
      }
    }
  }
}
```

```
Waiting_time[0] = 0;
current_time = Arrival_time[0] + Burst_time[0];
for (i = 1; i < n; i++){
    if (current_time < Arrival_time[i]){
        current_time = Arrival_time[i];
    }
    Waiting_time[i] = current_time - Arrival_time[i];
    current_time += Burst_time[i];
    total_waiting_time += Waiting_time[i];
}

printf("\n Process \t\t Arrival Time \t Burst Time \t
  Waiting Time \t Turnaround Time");
for(i=0; i<n; i++){
    Turn_around_time[i] = Burst_time[i] +
                          waiting_time[i];
total_turnaround_time += Turn_around_time[i];
printf("\nP[%d]\t\t%d \t\t %d \t\t %d \t\t %d",
process[i], Arrival_time[i], Burst_time[i],
Waiting_time[i], Turn_around_time[i]);
}

avg_waiting_time = (float) total_waiting_time/n;
avg_Turn_around_time = (float) total_turnaround
                               _time/n;

printf("\n Average Waiting Time: %.2f", avg_Waiting
                                        _time);
printf("\n Average Turnaround Time:
        %2.f\n", avg_Turn_around_time);
}
```

```
void SJF() {
int total_waiting_time = 0, total_turnaround_time = 0;
int completed = 0, current_time = 0, min_index;
int is_completed[20] = {0};
while (completed != n) {
    int min_burst_time = 9999;
    min_index = -1;
    for(i=0; i<n; i++){
        if (Arrival_time[i] <= current_time
&& is_completed[i] == 0) {
    if(Burst_time[i] < min_burst_time){
        min_burst_time = Burst_time[i];
        min_index = i;
    }
    if(Burst_time[i] == min_burst_time) {
        if(Arrival_time[i] < Arrival_time[min_index]
        {   min_burst_time = Burst_time[i];
            min_index = i;
        }
    }
  }
}
if(min_index != -1){
    Waiting_time[min_index] = current_time
        - Arrival_time[min_index];
current_time += Burst_time[min_index];
```

```
Turn_around_time[min_index] = current_time
                - Arrival_time[min_index];
total_waiting_time += Waiting_time[min_index];
total_turnaround_time += Turn_around_time
                [min_index];

is_completed[min_index] = 1;
    completed++;
} else {
    current_time++;
}
}
printf("\n Process \t\t Arrival Time \t Burst Time \t
 Waiting Time \t Turnaround Time");
for(i=0; i<n; i++) {
        printf("\n P[%d] \t\t %d \t\t %d \t\t %d
\t\t %d", process[i], Arrival_time[i], Burst_
time[i], Waiting_time[i], Turn_around_time[i]);
}
avg_Waiting_time = (float)total_waiting_time/n;
avg_Turn_around_time = (float) total_turnaround
_time /n;
printf("\n\n Average Waiting Time = %.2f",
    avg_Waiting_Time);
printf("\n Average Turnaround Time = %.2f\n"
    avg_ Turn_around_time);
}
```

```
int main(){
    printf("Enter total no of processes: ");
    Scanf("%d", &n);
    printf("Enter arrival Time & Burst Time:\n");
    for(i=0; i<n; i++){
        printf("P[%d] Arrival Time:", i+1);
        scanf("%d", &Arrival_time[i]);
        printf("P[%d] BurstTime:", i+1);
        Scanf("%d", &BurstTime[i]);
        process[i] = i+1;
    }
while(1){
    printf("\n ----MAIN MENU----\n");
    printf("1.FCFS \n2. SJF");
    printf("\nEnter your choice");
    Scanf("%d", &choice);
    switch(choice){
        case1: FCFS();
               break;
        Case 2: SJF();
               break;
        default: printf("Invalid Input!!!\n");
    }
}
return 0;
}
```

OUTPUT

Enter total no of processes : 4
Enter Arrival and Burst Time :
P[1] Arrival Time : 0
P[1] Burst Time : 3
P[2] AT : 1
P[2] BT : 6
P[3] AT : 4
P[3] BT : 4
P[4] AT : 6
P[4] BT : 2

--- MAIN MENU ---
1. FCFS
2. SJF
Enter your choice : 1

Process	Arrival Time	Burst Time	Waiting Time	Turnaround Time
P[1]	0	3	0	3
P[2]	1	6	2	8
P[3]	4	4	5	9
P[4]	6	2	7	9

Average Waiting Time : 3.5
Average Turnaround Time : 7.25

--- MAIN MENU ---
1. FCFS
2. SJF

Enter your choice : 2

process	Arrival Time	Burst Time	WaitingTime	Turnaround
P[1]	0	3	0	3
P[2]	1	6	2	8
P[3]	4	4	7	11
P[4]	6	2	3	5

Average waiting Time : 3

Average Turnaround Time : 6.75

15/5/24

a) Write a C program to simulate the following CPU Scheduling algorithm to find turn around time and waiting time

a) Priority (Non pre-emptive)

```
#include <stdio.h>
#include <stdlib.h>
struct process{
    int process_id;
    int burst_time;
    int priority;
    int waiting_time;
    int turnaround_time;
};
void find_average_time (struct process[ ], int);
void priority-scheduling (struct process[ ], int);
int main().
{
    int n, i;
    struct process proc[10];
    printf(" Enter no of processes: ");
    scanf (" %d", &n);
    for(i=0; i<n; i++)
    {   printf ("\nEnter process ID: ");
        Scanf(" %d", &proc[i].process_id);
        printf(" Enter burst time: ");
        scanf (" %d", &proc[i].burst_time);
```

```
Print ("Enter the priority");
Scanf ("%d", &proc[i].priority);
{
priority_scheduling (proc, n);
return 0;
}

void find_waiting_time (struct process proc[],
  int n, int wt[])
{ int i;
  wt[0] = 0;
  for (i=1; i<n; i++)
  {     wt[i] = proc[i-1].burst_time + wt[i-1];
  }
}

void find_turnaround_time (struct process proc[],
 int n, int wt[], int at[])
{    int i;
     for (i=0; i<n; i++)
     {
          tat[i] = proc[i].burst_time + wt[i];
     }
}
void find_average_time (struct process proc[],
  int n)
{  int wt[10], tat[10], total_wt=0, total_tat=0;
   find_waiting_time (proc, n, wt);
   find_turnaround_time (proc, n, wt, tat);
```

```
printf("\n Process ID\t Burst Time\t Priority\t
 Waiting Time\t Turnaround Time");
for (int i = 0; i < n; i++)
{   total_wt = total_wt + wt[i];
    total_tat = total_tat + tat[i];
    printf("\n %d\t\t %d\t\t %d\t\t %d\t\t
%d", proc[i].process_id, proc[i].burst_time,
proc[i].priority, wt[i], tat[i]);
}
printf("\n\n Average Waiting Time = %f",
(float)total_wt/n);
printf("\n Average Turnaround Time = %f\n",
(float)total_tat/n);
}
void priority_scheduling(struct process proc[],
int n)
{   int i, j, pos;
    Struct process temp;
    for (i=0; i<n; i++)
    {
        pos = i;
        for (j = i+1; j<n; j++)
        {   if (proc[j].priority < proc[pos].pr)
                pos = j;
        }
        temp = proc[i];
        proc[i] = proc[pos];
        proc[pos] = temp;
    }
    find_average_time (proc, n);
}
```

OUTPUT

Enter the no of processes: 5

Enter process ID: 1

Enter burst time: 4

Enter the priority: 2

PID: 2	PID: 3	PID: 4	PID: 5
BT: 3	BT: 1	BT: 5	BT: 2
P: 3	P: 4	P: 5	P: 5

Process ID	Burst Time	Priority	Waiting Time	Turnaround Time
1	4	2	0	4
2	3	3	4	7
3	1	4	7	8
4	5	5	8	13
5	2	5	13	15

b) Round Robin (Non-pre-emptive)

```
#include <stdio.h>
#include <stdbool.h>
int turnaroundtime (int process[], int n, int bt[]
 int wt[], int tat[]){
    for (int i=0; i<n; i++)
    tat[i] = bt[i] + wt[i];
    return 1;
}
int waitingtime (int processes[], int n, int bt[],
 int wt[], int quantum)
{   int rem_bt[n];
    for (int i=0; i<n; i++)
    rem_bt[i] = bt[i];
    int t=0;
    while (1)
    {  bool done = true;
       for (int i=0; i<n; i++)
       {  if (rem_bt[i] >0)
          {  done = false;
             if (rem_bt[i] > quantum)
             {   t+ = quantum;
                 rem_bt[i] - = quantum;
             }
             else { t = t + rem_bt[i];
                    wt[i] = t - bt[i];
                    rem_bt[i] = 0;
             }
       }
    }
}
```

```
    if (done == true)
      break;
  }
  return 1;
}

int findavgTime (int processes[], int n, int bt[],
  int quantum){
    int wt[n], tat[n], total_wt = 0, total_tat = 0;
    waitingtime (processes, n, bt, wt, quantum);
    turnaroundtime (processes, n, bt, wt, tat);
    printf ("\n\nProcesses \t\t Burst Time \t\t
Waiting Time \t\t turnaround time\n");
    for (int i=0; i<n; i++)
    {  total_wt = total_wt + wt[i];
       total_tat = total_tat + tat[i];
       printf ("\n \t %d \t\t %d \t\t %d\n",
i+1, bt[i], wt[i], tat[i]);
}
printf ("\nAverage waiting time = %f",
 (float) total_wt / (float) n);
printf ("\nAverage turnaround time = %f",
 (float) total_tat / (float) n);
  return 1;
}
```

```
int main()
{ int n, processes[n], burst_time[n], quantum;
  printf("Enter no of processes:");
  scanf("%d", &n);
  printf("\n Enter the quantum time:");
  scanf("%d", &quantum);
  int i=0;
  for(i=0; i<n; i++)
  { printf("\n Enter process:");
    scanf("%d", &processes[i]);
    printf("Enter Burst Time");
    scanf("%d", &burst_time[i]);
  }
findavgTime(processes, n, burst_time, quantum);
return 0;
}
```

OUTPUT :

Enter ~~process~~ no of processes: 3

Enter quantum time: 2

Enter process : 1 P: 2 P: 3

Enter burst Time: 5 BT: 7 BT: 3

Processes	Burst Time	Waiting Time	Turnaround time
1	5	7	12
2	7	8	15
3	3	8	11

5/6/24

5/6/24

Q Write a C program to stimulate real-time CPU Scheduling algorithms

a) Rate Monotonic

b) Earliest-deadline first

→
```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
#define max_process 10
typedef struct {
    int id;
    int burst_time;
    float priority;
} Task;
int num_of_process;
int execution_time[Max_process], period[max_pr
remain_time[Max_proc], deadline[Max_process],
remain_deadline[Max_process];

void get_process_info(int selected_algo)
{
    printf("Enter total no of process (max %d): ",
max); scanf("%d", &num_of_process);
if (num_of_process < 1)
{   exit(0); }
for (int i=0; i< no_of_process; i++)
{
```

```
printf("\n Process %d:\n", i+1);
printf(" ==> Execution time: ");
scanf("%d", &execution_time[i]);
remain_time[i] = execution_time[i];
if (selected_algo == 2)
{   printf(" ==> Deadline: ");
    scanf("%d", &deadline[i]);
} else
{   printf(" ==> Period: ");
    scanf("%d", &period[i]);
}
}
int max(int a, int b, int c)
{ int max;
  if (a>=b && a>=c)
    max = a;
  else if (b>=a && b>=c)
    max = b;
  else if (c>=a && c>=b)
    max = c;
  return max;
}
int get_observation_time(int selected_algo)
{   if (selected_algo == 1)
      { return max(period[0], period[1], period[2]);
    else if (selected_algo == 2)
        return max(deadline[0], deadline[1], deadline[2]);
}
```

```
void print_schedule ( int process_list[], int cycles)
{   printf ( "\n Scheduling: \n\n");
    printf ("Time");
    for (int i=0; i<cycles; i++)
    {   if (i<10)
            printf ("| 0%d", i);
        else
            printf ("| %d", i);
    }
    printf ("|\n");
    for( int i = 0; i < num_of_process; i++)
    {   printf ("P[%d]:", i+1);
        for (int j=0; j<cycles; j++)
        {   if ( process_list[j] == i+1)
                printf ("| ##### ");
            else
                printf("|      ");
        }
        printf ("|\n");
    }
}

void rate_monotonic (int time)
{   int process_list[100] = {0}, min=999,
    next_process = 0; float utilization = 0;
    for (int i=0; i< num_of_process; i++)
        utilization += (1.0 * execution_time[i])/
                        period[i];
```

```
int n = num_of_process;
int m = (float)(n * (pow(2, 1.0/n) - 1));
if (utilization > m)
{   printf("\n Given problem is not schedulable under
the said scheduling algo \n");
} for (int i = 0; i < time; i++)
{   min = 1000;
    for (int j = 0; j < num_of_process; j++)
    {   if (remain_time[j] > 0)
        {   if (min > period[j])
            {   min = period[j];
                next_process = j;
            }
        }
    }
    if (remain_time[next_process] > 0)
    {   process_list[i] = next_process + 1;
        remain_time[next_process] -= 1;
    }
    for (int k = 0; k < num_of_process; k++)
    {   if ((i+1) % period[k] == 0)
        {   remain_time[k] = execution_time[k];
            next_process = k;
        }
    }
}
print_Schedule(process_list, time);
}
```

```
void earlist_deadline_first (int time) {
    float utilization = 0;
    for (int i = 0; i < no_of_process; i++) {
        utilization += (1.0 * execution_time[i]) /
                        deadline[i];
    int n = num_of_process;
    int process [no_of_process];
    int max_deadline, current_process = 0, min_deadline,
    process_list [time];
    bool is_ready [num_of_process];
    for (int i=0; i<num_of_processes; i++) {
        if (deadline [i] > max_deadline)
            max_deadline = deadline [i];
    }
    for (int i=0; i< num_of_process; i++)
    { for (int j = i+1; j< num_of_process; j++)
        { if (deadline [j] < deadline [i])
            { int temp = execution_time[i];
              execution_time[j] = execution_time [i];
              execution_time [i] = temp;
              temp = deadline [i];
              deadline [j] = deadline [i];
              deadline [i] = temp;
              temp = process [j];
              process [j] = process [i];
              process [i] = temp;
            }
        }
    }
}
```

```
for (int i=0; i<num_of_process; i++)
{   remain_time[i] = execution_time[i];
    remain_deadline[i] = deadline[i];
}
for(int t=0; t<time; t++)
{   if(current_process!=-1)
    {--execution_time[current_process];
     process_list[t] = process[current_process];
}else
    process_list[t] = 0;
for(int i=0; i<num_of_process; i++){
    --deadline[i];
    if((execution_time[i]==0) && is_ready[i])
{   deadline[i] += remain_deadline[i];
    is_ready[i] = false;
}
if((deadline[i] <= remain_deadline[i]) &&
    (is_ready[i]==false))
{ execution_time[i] = remain_time[i];
  is_ready[i] = true;
}
}
min_deadline = max_deadline;
current_process = -1;
for(int i=0; i<num_of_process; i++)
{ if((deadline[i] <= min_deadline) &&
        (execution_time[i] > 0)) {
```

```
        current_process = i;
        min_deadline = deadline[i];
    }
  }
}
print_schedule(process_list, time);
}
int main()
{ int option;
  int observation_time;
  while (1) {
    printf("\n1. Rate Monotonic \n2. Earliest
Deadline first\n3. Exit");
  scanf("%d", &option);
  switch(option)
  {
    case 1: get_process_info(option);
            observation_time = get_observation
_time(option);
  rate_monotonic(observation_time);
  break;
  case 2: get_process_info(option);
    observation_time = get_observation_time(option);
    earliest_deadline_first(observation_time);
    break;
case 3: exit(0);
default:
    printf("\nInvalid statement\n");
  }
} return; }
```

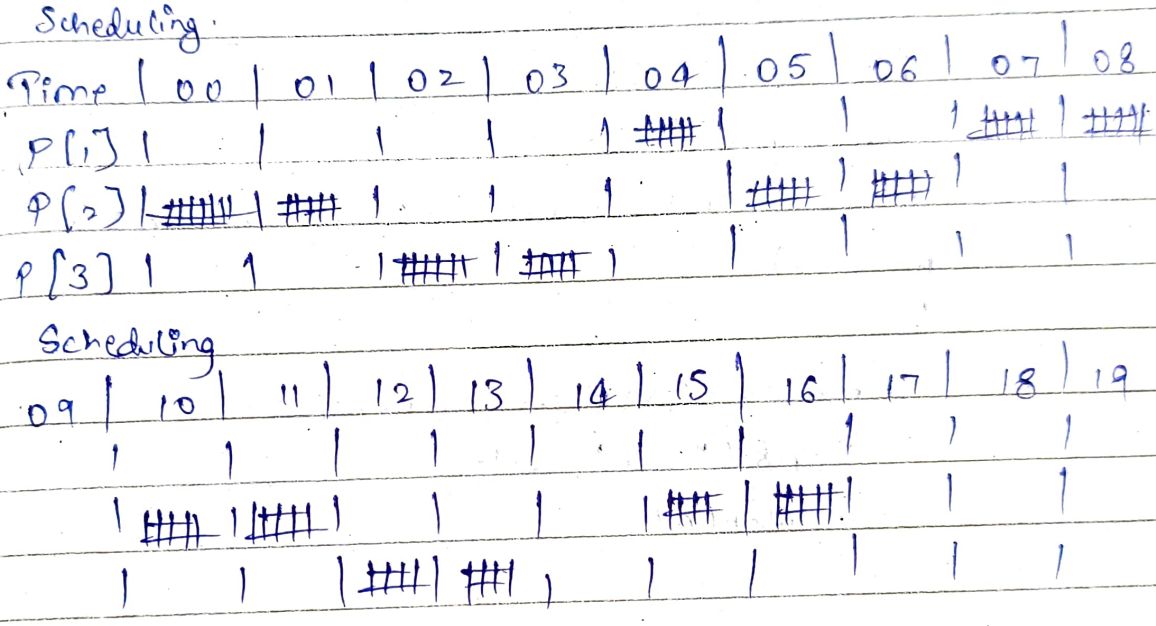
OUTPUT:

1. Rate Monotonic
2. Earliest Deadline first

Enter your choice: 1
Enter total no of process (max 10): 3

Process 1:	Process 2	Process 3
Execution time: 3	ET: 2	ET: 2
Deadline: 20	D: 5	D: 10

Scheduling:



Scheduling

Enter your choice: 2
Enter total no of processes (max 10): 3

Process 1:	Process 2	Process 3
Execution time: 1	ET: 1	ET: 2
Deadline: 3	D: 4	D: 8

Scheduling: