

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <limits.h>
4
5 //working
6 void printArray(int rows, int cols, int** array) {
7     for (int i = 0; i < rows; i++) {
8         for (int j = 0; j < cols; j++) {
9             printf("%d\t", array[i][j]);
10        }
11        printf("\n");
12    }
13 }
14
15 //working
16 int** input(int* rows, int cols) {
17     cols = 4;
18     printf("This will take input row wise.\n");
19
20     printf("The usual format is: \n Process #\tArrival time\tBurst time\tCompletion time\n");
21     printf("Enter the number of rows (Processes): ");
22     scanf("%d", rows);
23     //Dynamic memory allocation is when the data structure is changed at runtime.
24     int** array = (int**)malloc(*rows * sizeof(int*)); //refer to malloc.c
25     for (int i = 0; i < *rows; i++) {
26         array[i] = (int*)malloc(cols * sizeof(int));
27     }
28
29     printf("Enter the arrival and burst times:\n");
30     for (int i = 0; i < *rows; i++) {
31         array[i][0] = i;
32
33         printf("Process %d Arrival time: ", i);
34         scanf("%d", &array[i][1]);
35
36         printf("Process %d Burst time: ", i);
37         scanf("%d", &array[i][2]);
38
39         array[i][3] = 0; //initialize completion time to 0
40     }
41
42     printf("Original array: \nPID\tAT\tBT\tCT\n");
43
44     printArray(*rows, cols, array);
45
46     return array;
47 }
48
49 // Not workig: row int ptr issue
50 void sortRowsBySecondColumn(int** array, int rows) {
51     for (int i = 0; i < rows - 1; i++) {
52         for (int j = 0; j < rows - i - 1; j++) {
53             if (array[j][1] > array[j + 1][1]){
54                 for (int k = 0; k < 4; k++) {
55                     int temp = array[j][k];
56                     array[j][k] = array[j + 1][k];
57                     array[j + 1][k] = temp;
58                 }
59             }
60         }
61     }
62 }
63
64 //working : manage how to pass an array here
65 void ganttChart(int* arr, int length){
66     //top:
67     for(int i = 0; i < length; i++){
68         for(int j = 0; j < arr[i]; j++){
69             printf("_");
70         }
71     }
72     printf("\n");
73
74     //inside block:
75     for(int i = 0; i < length; i++){
76         printf("|");

```

```

77     for(int j = 0; j < arr[i]; j++){
78         if(j == arr[i]/2){
79             printf("%d ", arr[i]);
80             if(arr[i] >= 10){ //for two digit numbers
81                 j++;
82             }
83         }
84         else{
85             printf(" ");
86         }
87     }
88 }
89 printf("\n");
90
91 //bottom:
92 for(int i = 0; i < length; i++){
93     printf("|");
94     for(int j = 0; j < arr[i]; j++){
95         printf("_");
96     }
97 }
98 printf("\n");
99 }
100
101 //Not working, chnge it from 2D array to a ptr
102 void nonPreemption(){
103     // Logic : sort the array according to arrival time. Then, directly display the order in which they are.
104     int rows;
105     int cols = 4;
106     int** matrix = input(&rows, cols);
107
108     sortRowsBySecondColumn(matrix, rows);
109
110     //printing the sorted array
111     printf("Sorted array by Arrival Time (AT):\n");
112     printArray(rows, cols, matrix);
113
114     int currentTime = 0;
115     for(int i = 0; i < rows; i++){
116         if(currentTime < matrix[i][1]){
117             currentTime = matrix[i][1];
118         }
119         currentTime += matrix[i][2];
120         matrix[i][3] = currentTime; // update completion time
121     }
122
123     printf("Final order with which processes run: ");
124     for(int i = 0; i < rows; i++) {
125         printf("%d --> ", matrix[i][0]);
126     }
127     printf("END\n");
128
129     // Free allocated memory
130     for (int i = 0; i < rows; i++) {
131         free(matrix[i]);
132     }
133     free(matrix);
134 }
135
136 void preemptionSRTF() {
137     int rows;
138     int cols = 4;
139     int** matrix = input(&rows, cols);
140
141     int* remainingTime = (int*)malloc(rows * sizeof(int));
142     int* completionTime = (int*)malloc(rows * sizeof(int));
143     int* startTime = (int*)malloc(rows * sizeof(int));
144     int* waitingTime = (int*)malloc(rows * sizeof(int));
145
146     for (int i = 0; i < rows; i++) {
147         remainingTime[i] = matrix[i][2]; // Initialize remaining time with burst time
148         completionTime[i] = 0;
149         startTime[i] = 0;
150         waitingTime[i] = 0;
151     }
152
153     int completed = 0, currentTime = 0, minBurstTime = INT_MAX;

```

```

153     int shortest = 0, finishTime;
154     int check = 0;
155
156     while (completed != rows) {
157         for (int j = 0; j < rows; j++) {
158             if ((matrix[j][1] <= currentTime) && (remainingTime[j] < minBurstTime) && remainingTime[j] > 0) {
159                 minBurstTime = remainingTime[j];
160                 shortest = j;
161                 check = 1;
162             }
163         }
164
165         if (check == 0) {
166             currentTime++;
167             continue;
168         }
169
170         remainingTime[shortest]--;
171
172         minBurstTime = remainingTime[shortest];
173         if (minBurstTime == 0) {
174             minBurstTime = INT_MAX;
175         }
176
177         if (remainingTime[shortest] == 0) {
178             completed++;
179             check = 0;
180
181             finishTime = currentTime + 1;
182             completionTime[shortest] = finishTime;
183             startTime[shortest] = finishTime - matrix[shortest][2] - matrix[shortest][1];
184             if (startTime[shortest] < 0) {
185                 startTime[shortest] = 0;
186             }
187         }
188         currentTime++;
189     }
190
191     printf("Final order with which processes run: ");
192     for (int i = 0; i < rows; i++) {
193         printf("%d --> ", matrix[i][0]);
194     }
195     printf("END\n");
196
197     printf("Process completion times:\nPID\tTAT\tBT\tCT\tTAT\tWT\n");
198     for (int i = 0; i < rows; i++) {
199         int tat = completionTime[i] - matrix[i][1];
200         int wt = tat - matrix[i][2];
201         printf("%d\t%d\t%d\t%d\t%d\t%d\n", matrix[i][0], matrix[i][1], matrix[i][2], completionTime[i], tat, wt);
202     }
203
204     //free allocated memory
205     for (int i = 0; i < rows; i++) {
206         free(matrix[i]);
207     }
208     free(matrix);
209     free(remainingTime);
210     free(completionTime);
211     free(startTime);
212     free(waitingTime);
213 }
214
215 void roundRobin(int** matrix, int rows, int quantum) {
216     int* remainingTime = (int*)malloc(rows * sizeof(int));
217     int* completionTime = (int*)malloc(rows * sizeof(int));
218     int* waitingTime = (int*)malloc(rows * sizeof(int));
219     int* turnaroundTime = (int*)malloc(rows * sizeof(int));
220
221     for (int i = 0; i < rows; i++) {
222         remainingTime[i] = matrix[i][2]; // Initialize remaining time with burst time
223         completionTime[i] = 0;
224         waitingTime[i] = 0;
225         turnaroundTime[i] = 0;
226     }
227
228     int currentTime = 0;
229     int completed = 0;

```

```

230     int completed = 0;
231     int index = 0;
232
233     while (completed != rows) {
234         int done = 1;
235         for (int i = 0; i < rows; i++) {
236             if (remainingTime[i] > 0) {
237                 done = 0;
238                 if (remainingTime[i] > quantum) {
239                     currentTime += quantum;
240                     remainingTime[i] -= quantum;
241                 } else {
242                     currentTime += remainingTime[i];
243                     completionTime[i] = currentTime;
244                     remainingTime[i] = 0;
245                     completed++;
246                 }
247             }
248         }
249         if (done == 1) {
250             break;
251         }
252     }
253
254     printf("Final order with which processes run: ");
255     for (int i = 0; i < rows; i++) {
256         printf("%d --> ", matrix[i][0]);
257     }
258     printf("END\n");
259
260     printf("Process completion times:\nPID\tAT\tBT\tCT\tTAT\tWT\n");
261     for (int i = 0; i < rows; i++) {
262         turnaroundTime[i] = completionTime[i] - matrix[i][1];
263         waitingTime[i] = turnaroundTime[i] - matrix[i][2];
264         printf("%d\t%d\t%d\t%d\t%d\t%d\n", matrix[i][0], matrix[i][1], matrix[i][2], completionTime[i], turnaroundT
265     }
266
267     //free allocated memory
268     free(remainingTime);
269     free(completionTime);
270     free(waitingTime);
271     free(turnaroundTime);
272 }
273
274
275 int main() {
276     nonPreemption();
277     //int array[] = {5,3,7,2};
278     //int length = sizeof(array) / sizeof(array[0]);
279     // ganttChart(array,length);
280     //int rows, cols;
281     //int** processArray = input(&rows, cols);
282
283     //printArray(rows, cols, processArray);
284
285     return 0;
286 }

```

• tanishkhot@Tanishs-MacBook-Air A3 % ./a.out

This will take input row wise.

The usual format is:

Process # : Arrival time : Burst time : Completion time

Enter the number of rows (Processes): 4

Enter the arrival and burst times:

Process 0 Arrival time: 2

Process 0 Burst time: 5

Process 1 Arrival time: 1

Process 1 Burst time: 3

Process 2 Arrival time: 4

Process 2 Burst time: 7

Process 3 Arrival time: 0

Process 3 Burst time: 2

Original array:

PID	AT	BT	CT
0	2	5	0
1	1	3	0
2	4	7	0
3	0	2	0

Sorted array by Arrival Time (AT):

3	0	2	0
1	1	3	0
0	2	5	0
2	4	7	0

Final order with which processes run: 3 --> 1 --> 0 --> 2 --> END