General Instructions

- This lab test carries 20 marks. The test consists of two questions numbered 1 and 2.
- Programs should be written in *C* language.
- Assume that all inputs are valid.
- Sample inputs are just indicative.
- Use of global variables is NOT permitted.
- The input should be read from, and the output should be printed to, the console.
- **No clarifications regarding questions will be entertained. If there is any missing data you may make appropriate assumptions and write the assumptions clearly in the design sheet.**
- Solve Part 2 only after submitting a solution (design and code) for Part 1.
- **The students must start with the design of Part 1 and upload the design of Part 1 in the EduServer before 3:30 pm.**
- After uploading the design, students can begin the implementation of Part 1. The source file of Part 1 should be uploaded in the EduServer before 5:00 pm.
- There will be a viva voce during the test.
- Design Submission:
  1. Read the question, understand the problem and write the design (in a sheet of paper) for the indicated function(s) as algorithm(s)(in pseudocode), as per the given prototype.
  2. Take a clear photograph of the handwritten design sheet and submit through the link in eduserver.
  3. The design must be written using pseudocode conventions. There will be a reduction in marks if the student writes C code instead of pseudocode.
  4. For Part 2 also, upload the design first and then start the implementation. Students can upload Part 2 files (design/implementation) till 5:30 pm.
- **The implementation *must be* completely based on the design already submitted.**
- The source code file should be named in the format

  `TEST<NUMBER>_<ROLLNO>_<FIRST-NAME>_<PROGRAM-NUMBER>.c`

  (For example, TEST1_B190001CS_LAXMAN_1.c)

  The source file must be zipped and uploaded. The name of the zip file must be

  `TEST<NUMBER>_<ROLLNO>_<FIRST-NAME>.zip`

  (For example: TEST1_ B190001CS_LAXMAN.zip)

Mark distribution

Maximum marks – 20

- Question 1: 14 Marks (Design - 7 marks, Implementation and Test cases - 5 marks, Viva voce - 2 marks)
- Question 2: 6 Marks (Design - 3 marks, Implementation and Test cases - 3 marks)

1. A set of n processes, with process IDs numbered from 0 to $n-1$, need to be run on a computer. Each process has an associated *processing time*, which specifies the number of seconds that it would need to run for. After every t seconds(*the maximum time over which a process is continuously run*), the computer puts the current process on wait and runs the next waiting process. The computer runs in this manner until all the processes have finished execution.

   Write a C program that implements the following functions as per the function prototypes given below. (You have to store the processing time information of the processes in an array A of size n. Similarly, you have to store the completion time information of the processes in an array B of size n).

   - $read\_and\_store(A, n)$ – For each of the n processes, read its processing time from the console and store it in the array A.
   - $run\_processes(A, B, n, t)$ - Run all the processes, by following the specifications given below.
     1. The processes are run in the order of their process IDs.
     2. At time 1, the first process (which has process id 0) is run.
     3. After every t seconds, put the current process on wait and run the next waiting process. (You need not use standard time functions, just decrement the remaining processing time of the current process by t)
     4. When the execution of a process is completed, store it's time of completion in the array B at the appropriate index.

     Please refer the figure below for an illustrative example.
   - $list\_process(B, n)$ – Print the contents of the array B, with the elements separated by a single space.

   Example

   The number of processes, $n = 4$ and $t = 3$

Process ID	Processing Time
0	5
1	5
2	2
3	3

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Process	0	0	0	1	1	1	2	2	-	3	3	3	0	0	-	1	1	-
Process completed								2				3		0			1	

   Input Format

   The first line contains an integer n which is the number of processes.

   The second line contains the processing times of the n processes as n space-separated integers, in the order of their process IDs.

The third line contains an integer t, the maximum time over which a process is continuously run.

Output Format

Using the $list\_process()$ function, print the time of completion of each process in the order of their process IDs, separated by single spaces.

Sample Input and Output

Input

```
4
5 5 2 3
3
```

Output

```
14 17 8 12
```

2. Suppose that, in the previous question, every process also has an *arrival time* associated with it, in addition to the processing time. The computer should run the processes, as described below.

   1. Initially, consider the processes which have arrived at time 0. Out of them, run the process p which has the smallest processing time.
   2. When the time slot t assigned to p is over or the execution of p is completed, the next process to run should be selected as follows.
      (a) At time i, consider the processes which have arrived at or before time i.
      (b) Out of these, consider the processes that have the minimum processing time remaining.
      (c) Out of these, consider the processes which arrived first.
      (d) Out of these, run the process with the smallest process ID.

   Write a C program that implements the following functions. (You can decide the function prototypes based on your design.)

   - $read()$ – Reads and stores the details of the n processes.
   - $run\_processes()$ – Runs all the processes by following the specifications given above.
   - $list\_process()$ – Print the process ID and finishing time of each of the n processes, as detailed in the Output Format section.

Input Format

The first line contains an integer n which is the number of processes.

The second line contains an integer t, the maximum time over which any process is continuously run.

Each of the next n lines contains two integers corresponding to the *arrival time* and *processing time* of a process, separated by a single space, in the order of their process IDs.

Output Format

The output should contain n lines, corresponding to the n processes and ordered by their completion times. Each line of the output should contain the process ID and completion time of the corresponding processes, separated by a single space.

Sample Input and Output

Input

```
6
5
0 11
4 5
7 3
9 7
12 4
14 7
```

Output

```
1 10
2 13
4 17
0 23
3 30
5 37
```