**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 doctor is assigned to a set of $n$ wards, numbered from 1 to $n$, each having a certain number of patients. In each of his rounds, after every $t$ minutes (the time that the doctor spends in each ward), the doctor moves on to the next ward. He continues to take rounds until he has examined all the patients of the $n$ wards. Also, for each ward, there is a specified *inspection time* that the doctor would need to examine all the patients in that ward.

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

   - $get\_inspection\_time(n, D)$ - For each of the $n$ wards, read its inspection time from the console and store it in the array $D$.
   - $visit\_ward(D, C, n, t)$ - Determine the order in which the doctor visits the wards, by following the specifications given below.
     1. The doctor visits each ward in the order of the ward IDs.
     2. At time 1, the doctor visits the first ward (which has ward ID 1).
     3. After every $t$ minutes, the doctor visits the next ward.
     4. When all the patients in a ward are examined, store the time of completion of inspection in array $C$ in the corresponding index.

     *Please refer the figure below for an illustrative example.*
   - $display(C, n)$ – Print the contents of the array $C$ of size $n$, with the elements separated by a single space.

**Example**

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

| Ward ID | Inspection Time |
|---|---|
| 1 | 7 |
| 2 | 3 |
| 3 | 8 |

| Time | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ward ID | 1 | 1 | 1 | 1 | 2 | 2 | 2 | - | 3 | 3 | 3 | 3 | 1 | 1 | 1 | - | 3 | 3 | 3 | 3 |
| Ward completed | | | | | | | 2 | | | | | | | | 1 | | | | | 3 |

**Input Format**

The first line contains an integer $n$, the number of wards.

The second line contains $n$ space-separated integers, the inspection times of the $n$ wards, in the order of their ward IDs.

The third line contains an integer $t$, the time that the doctor spends in each ward.

**Output Format**

Using the $display()$ function, print the time of completion of the inspection of each ward in the order of their ward IDs, separated by single spaces.

**Sample Input and Output**

**Input**

```
3
7 3 8
4
```

**Output**

```
15 7 20
```

2. Suppose that, in the previous question, the doctor would only visit a ward after the ward's attender has informed him that they are ready for inspection. That is, every ward also has an *ready time* associated with it, in addition to the inspection time. The doctor should visit the wards, as described below.

   1. Initially, consider the wards which are ready at time $0$. Out of them, the doctor visits the ward $w$ which has the smallest inspection time.
   2. After spending $t$ minutes in the current ward or once every patient in that ward has been examined, the doctor moves on to the next ward, as described below.
      (a) At time $i$, consider the wards which are ready at or before time $i$.
      (b) Out of these, consider the wards that have the minimum inspection time remaining.
      (c) Out of these, consider the wards which were ready.
      (d) Out of these, the doctor visits the ward with the smallest ward ID.

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

   - $get\_times()$ – Reads and stores the inspection times and ready times of the $n$ wards.
   - $visit\_wards()$ – The doctor visits all the wards by following the specifications given above.
   - $ward\_list()$ – Print the ward ID and time of completion of inspections for each of the $n$ wards, as detailed in the Output Format section.

**Input Format**

The first line contains an integer $n$, the number of wards.

The second line contains an integer $t$, the maximum time the doctor spends in any ward before moving on.

Each of the next $n$ lines contains two integers corresponding to the *ready time* and *inspection time* of a ward, separated by a single space, in the order of their ward IDs.

**Output Format**

The output should contain $n$ lines, corresponding to the $n$ wards and ordered by the times of completion of the inspections in those wards. Each line of the output should contain the ward ID and completion time of the corresponding wards, separated by a single space.

**Sample Input and Output**

**Input**

```
6
5
0 10
3 5
6 6
```

```
7 7
9 5
10 2
```

**Output**

```
1 10
6 12
2 17
5 22
3 28
4 35
```