General Instructions

- This lab test carries 10 marks.
- Program should be written in C language.
- Assume that all inputs are valid.
- Sample inputs are just indicative and need not handle every input case.
- Use of global variables is NOT permitted.
- The function prototypes given in the questions should not be changed.
- 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.**
- **Students have to first write the design and then only proceed with the implementation.**
- There will be a viva voce during the test.
- Design submission:
  1. Read the question, understand the problem and write the design in pseudocode (in the shared Google document) for the indicated functions only. You are advised to **properly think about the solution before starting to write your design** to avoid wasting your time on rewrites.
  2. The design written should **clearly** convey your overall idea for the solution; the programming specific details may be changed over the course of the implementation. There will be a reduction in marks if the student writes C code instead of pseudocode.
  3. The edit permission of the shared document for writing the design will be revoked at 3:05 PM.
  4. Upload the pdf of the shared google document for writing the design in EduServer before 3:10 PM. **Only the designs uploaded in Eduserver will be considered for evaluation.**
  5. Once designed, you should **write a program that implements your design**.
  6. If, while implementing, you realize that your design has major issues, you can ask your evaluator for permission to change your design. The evaluator will decide whether to permit the modification or not by looking at your current progress.
  7. In any case, modifications to the design will not be permitted beyond 3:45 PM.
- Mode of submission and timings:
  1. Design (upload the pdf of the shared google document in EduServer) - 3:10 PM
  2. Implementation (upload the source file in EduServer) - 5:10 PM

- While implementing, you may use the source codes that you have previously submitted for the assignments, if you feel that it will be helpful.
- There will be some test cases that only test the correctness of specific functions and not the entire program. As such, if you are not able to complete your program, you should still **make sure that your submitted code will compile and run without errors** to get the marks for such test cases.
- The source code file should be named in the format

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

  (For example, TEST3_B190001CS_LAXMAN_1.c)

  The source file must be zipped and uploaded. Only zip files may be uploaded, even if they contain on a single .c file. The name of the zip file must be

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

  (For example: TEST3_ B190001CS_LAXMAN.zip)
- Naming conventions **must** be strictly followed. Any deviations from the specified conventions, or associated requests after the exam, will not be considered.
- Any malpractice observed will lead to zero marks in the test. These will also be reported to the department for permission to award F grade in the course.

Mark distribution

Maximum marks – 10

- Design - 4 marks, Implementation and Test cases - 4 marks, Viva voce - 2 marks

1. Operating Systems maintain a data structure called Process Control Block(PCB) for each process. Each process has a *process_id* and *state*. The *process_id*s are numbered from 1 and are distinct. A process can be in any one of the following states: 'new', '$ready$', '$running$', '$waiting$', or '$terminated$'. For simplicity, we assume that only the *process_id* of a process is stored in the PCB (*state* information is not stored in the PCB).

   Write a menu-driven program to maintain PCBs of the given processes using *Integer Arrays*. Maintain four different arrays as described below:

   - New (denoted by N), $Ready$ (denoted by R) and $Waiting$ (denoted by W): Stores the *process_id*s of the processes in the corresponding states in the order in which the requests were received.
   - $Terminated$ (denoted by T): Stores the *process_id*s of the $terminated$ processes in their increasing order.

   Your program must implement the following functions.

   - $main()$: Repeatedly read a character 'c', 'l', 'u', 's' or 't' from console and call the sub-functions appropriately until character 'e' is encountered.
   - LIST_PROCESSES(A, n): List the *process_id*s of all the processes in the array A of length n. If the array A is empty, then print -1.
   - UPDATE_STATE(k, A_1, A_2, n_1, n_2): Move the*process_id* k from the array A_1 of length n_1 to the array A_2 of length n_2, maintaining the properties of both the arrays. The state transitions to be performed by the UPDATE_STATE() function are limited to: *new* $\rightarrow$ *ready* and *waiting* $\rightarrow$ *ready*. Other valid state transitions are handled separately (Refer the characters 'u', 's'and 't'in the input format).
   - SCHEDULE(p, R, n): Scheduling is to be done whenever the state of the currently running process is updated, or an explicit schedule command('s') is received. Given the *process_id* p of the currently running process and the Ready array R of length n, scheduling of the the next process is done as follows.
     1. At any instance, at most *one* process can be run.
     2. If the Ready array R is empty then print -1 and return.
     3. Otherwise, from the Ready array R, select the *process_id* of the process that was ready first.
     4. If there is a process currently running, move p into the Ready array R.
     5. Let i be the *process_id* selected in step 3. Remove i from the Ready array R and set it as the currently running process.

Design Instructions

- Write the design for the functions $main()$, UPDATE_STATE() and SCHEDULE() only.
- While passing the array length as function parameter, use **pass by reference** method.

Input Format

The input consists of multiple lines. Each line may contain a character from 'c', 'l', 'u', 's', 't', 'e' followed by at most one integer and/or two strings. The integers, if given, are in the range $[1, 10^6]$.

- Character 'c' : Select the next available *process_id* for the new process and insert it into the New array N.
- Character 'l' is followed by an integer from $\{1, 2, 3, 4\}$.
  - If character l is followed by 1, print all the *process_id*s in the array N, separated by a space.
  - If character l is followed by 2, print all the *process_id*s in the array R, separated by a space.
  - If character l is followed by 3, print all the *process_id*s in the array W, separated by a space.
  - If character l is followed by 4, print all the *process_id*s in the array T, separated by a space.
- Character 'u' followed by an integer k and two strings $s1$ and $s2$, where k is the *process_id* of the process whose state is to be changed from $s1$ to $s2$.
  - If an update request is received to change the state of a process with *process_id* = k from '$running$' to '$waiting$',
    1. Move the *process_id* k into the Waiting array W.
    2. Schedule the next process from the Ready array R using the function SCHEDULE().
  - For any other update request, update the two arrays corresponding to states $s1$ and $s2$ using function UPDATE_STATE(). The state transitions to be performed by the UPDATE_STATE() function are limited to: *new* $\rightarrow$ *ready* and *waiting* $\rightarrow$ *ready*.
- Character 's': Schedule the next process from the Ready array R using the function SCHEDULE().
- Character 't': Print the *process_id* of the currently running process (assume there is always a running process when 't'is encountered), and move it to the Terminated array T. Also schedule the next process from the Ready array R using the function SCHEDULE().
- Character 'e' : End the program.

Output Format:

- The output (if any) of each command should be printed in a separate line.

Sample Input :

```
c
c
c
c
c
c
l 1
u 3 new ready
u 6 new ready
u 2 new ready
l 2
s
u 3 running waiting
t
s
u 1 new ready
c
l 1
u 5 new ready
s
t
u 3 waiting ready
t
l 3
l 1
l 2
l 4
e
```

Output:

```
1 2 3 4 5 6
3 6 2
6
-1
4 5 7
1
5
-1
4 7
3
1 5 6
```