General Instructions

- This lab test carries 20 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 – 20

- Design - 10 marks, Implementation and Test cases - 8 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*, *priority* and *state*. The *process_id*s are numbered from 1 and are distinct. The *priority* of a process can be $high$, $medium$ or low. 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* and *priority* of a process are 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 *Singly linked lists*. Maintain four different lists as described below:

   - New List (denoted by N): Stores the newly created PCBs in the order in which they were created.
   - Ready List (denoted by R): Stores the PCBs of the processes that are ready to be scheduled.
     - The PCBs with higher priorities should occur in the list before the ones with lower priorities.
     - If more than one PCB has the same *priority*, store them in the order in which they got ready.
   - Waiting List (denoted by W): Stores the PCBs of the processes that are in $waiting$ state, in the order in which they were requested to wait.
   - Terminated List (denoted by T): Stores the PCBs of the $terminated$ processes in the increasing order of their *process_id*.

   A new process is assigned the smallest available *process_id* as its PCB gets created. The *process_id* of a terminated process should be reused for the next new process. When the *process_id* of a $terminated$ process t is reassigned for a new process, set the *process_id* of t to 999, and place it at the end of the list T.

   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.
   - CREATE_PCB(k, p): Create and return a pointer to the new PCB with *process_id* = k and *priority* = p. Assign the pointer $next$ of the PCB to NIL.
   - LIST_PROCESSES(L): List the *process_id*s of all the PCBs in L. If the list L is empty, then print -1.
   - UPDATE_STATE(k, L_1, L_2): Move the PCB with *process_id* = k from the list L_1 to the list L_2, maintaining the properties of both the lists. 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 commands 'u', 's'and 't'in the input format).
   - SCHEDULE(p, R): 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 PCB p of the currently running process and the Ready List R, scheduling of the next process is done as follows.
     1. At any instance, at most *one* process can be run.
     2. If Ready List R is empty then print -1 and return.
     3. Otherwise, select the PCB with the highest priority from the ready list. If more than one process has the same priority, select the process that was ready first.
     4. If there is a process p currently running, move p into the Ready List R.

5. Let i be the PCB of the process selected in step 3. Remove i from the ready list R and set it as the currently running process.

Design Instructions

- Write the design for the functions $main()$, UPDATE_STATE() and SCHEDULE() only.
- Perform the insertion of a process into the New List and the Ready List in **constant time**. Efficiency in design carries weightage.
- When a PCB is moved from one list to another, the PCB should not be physically deleted, only the link updation must be done.

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 $[1, 10^6]$ and the string is from the set {*new*, *ready*, *waiting*, *running*, *high*, *medium*, *low*}.

- Character *'c'* is followed by a string: Create a new PCB with the smallest available *process_id* and the given string as *priority*. Insert the PCB to the New List N.
- Character *'l'* is followed by an integer from $\{1, 2, 3, 4\}$.
  - If character l is followed by 1, print the *process_id*s of all the PCBs in the New List N, separated by a space.
  - If character l is followed by 2, print the *process_id*s of all the PCBs in the Ready List R, separated by a space.
  - If character l is followed by 3, print the *process_id*s of all the PCBs in the Waiting List W, separated by a space.
  - If character l is followed by 4, print the *process_id*s of all the PCBs in the Terminated List T, separated by a space.
- Character *'u'* is followed by an integer k and two strings $s1$ and $s2$, where k is the *process_id* of the PCB whose state is to be changed from $s1$ to $s2$:
  - If an update request is received to change the state of a PCB with *process_id* = k from *'running'*to *'waiting'*,
    1. Move that process into the Waiting List W.
    2. Schedule the next process from the Ready List R using the function SCHEDULE().
  - For any other update request, update the two lists 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* → *ready* and *waiting* → *ready*.
- Character *'s'*: Schedule the next process from the Ready List R using the function SCHEDULE().
- Character *'t'*: Print the *process_id* of the currently running process p (assume there is always a running process when *'t'*is encountered), and move it to the Terminated List T. Also schedule the next process from the Ready List 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 medium
c high
c high
c low
c low
c medium
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 low
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 2 6
2
-1
4 5 2
1
6
-1
4 2
5
1 6 999
```