Introduction
One technique for dealing with deadlock is called “detect and recover.” In this scheme, some procedure is used to identify when a deadlock occurs, and then another procedure is used to deal with the blocked processes. One technique to identify a deadlock is to maintain a resource graph that identifies all processes, all resources, and the relationships between them (that is, which processes exclusively own which resources, and which processes are blocked waiting for which resources), and check for cycles in the resource graph. For this assignment, you will write a program that simulates the execution of multiple processes with respect to their resource allocation and resource release requests, and detect when (or if) deadlock occurs. If deadlock occurs, your program will then display a list of the processes and resources involved in the deadlock.

To simulate the behavior of the processes in the system, your program will use input that describes the sequence of actions each process takes to lock, release (unlock), and use resources. Each such action consists of a capital letter L, U or C respectively, immediately followed by a non-negative integer n; the interpretation of n depends on the particular action. The various actions permitted are as follows.

- If the action is L, the process (call it p) that performs the action requests exclusive use of the resource identified by n. The nr resources in the system are identified by integers in the range 1 to nr. If the resource is available (that is, not currently locked by any other process) when process p requests its use, then process p is given exclusive use of the resource and is allowed to continue. If the resource is not available, process p is blocked and placed at the end of the first-in-first-out queue of processes waiting on the resource. Process p remains on the queue until the resource becomes available and it (process p) is at the head of the queue of waiting processes. Thus the first process to block while requesting use of a resource is the process that is moved to the ready state when the resource again becomes available.

- If the action is U, process p relinquishes its exclusive use of the resource identified by n. This action will also move a single blocked process (say process q) from the head of the queue of processes waiting on this resource to the end of the system’s ready queue. Note that immediately after this waiting process (q) moves to the end of the ready queue, the process that just released the resource (p) will be moved to the end of the ready queue, just after the process that was at the head of the waiting queue (q).

- If the action is C, then the process “computes” for n units of time, maintaining exclusive use of any resources it may have previously obtained but not yet released. n will never be smaller than 1; that is, there will never be any vacuous “compute” actions in a process. Of course, the “computation” is really fictional, and will just cause the simulated system time to increase by n.

A process terminates after it has successfully executed each of the actions specified for it in the input data. Note that we make the (unusual) assumption that if a process terminates still having exclusive use of one or more resources, then those resources are forever unavailable (at least until the next simulation)!
For example, suppose a process is described by the following sequence of actions:

\[
L1 \hspace{0.1cm} L2 \hspace{0.1cm} C10 \hspace{0.1cm} U1 \hspace{0.1cm} U2
\]

This process will first request exclusive use of resource 1, then exclusive use of resource 2. Next it will compute for 10 units of time (maintaining exclusive use of resources 1 and 2), after which it will relinquish its use of resources 1 and 2 (in that order).

There will be no more than 50 processes in the system in any simulation, and we will assume there is just a single CPU. Each process will have its own sequence of actions describing its execution, and there will be no more than 50 actions for any process. There will be no more than 50 resources in any simulation.

Each successful request for the exclusive use (L) or release (U) of a resource by a process requires one time unit to complete, and a compute action (C) requires the number of time units specified by \( n \) for that action. The example shown above will require a total of 14 time units to complete. Since all processes are assumed to be in the ready queue (in increasing process number order: 1, 2, \( \ldots \), \( np \)) at the beginning of a simulation, and the first time unit is numbered 0, the process in the example above will terminate at time 14.

The execution of the sequences describing each process is done in a round-robin manner. One unit of time is given to each process, in turn, starting with process 1, then process 2, \( \ldots \), and then process \( np \). Execution then continues with process 1 again (assuming it didn’t block or terminate during its previous execution). Of course, when a process completes, it is no longer given any units of execution time. If a process becomes blocked, it does not receive any units of execution time until it is unblocked. If a process \( p \) requests a resource \( r \), but is not able to immediately obtain exclusive use of \( r \) (because another process \( q \) is using it), then process \( p \) is blocked while waiting for the resource \( r \) to become available. When process \( q \) releases resource \( r \), the first process waiting for the resource (that is, the first process in the queue of processes waiting for resource \( r \)) is moved to the end of the queue of ready processes. Note that this does not guarantee that process \( p \) will successfully obtain the resource when it next runs, since another ready process (ahead of \( p \) in the ready queue) may run and request exclusive use of the resource before \( p \) is selected for execution.

Let’s look at a complete example. Suppose we have three processes with execution action sequences as follows:

- Process 1: \( L1 \hspace{0.1cm} L2 \hspace{0.1cm} C2 \hspace{0.1cm} U1 \hspace{0.1cm} U2 \)
- Process 2: \( L1 \hspace{0.1cm} L2 \hspace{0.1cm} C2 \hspace{0.1cm} U1 \hspace{0.1cm} U2 \)
- Process 3: \( L3 \hspace{0.1cm} C5 \hspace{0.1cm} U3 \hspace{0.1cm} C2 \)

Simulation 2 in the sample input and output (below) shows the sequence in which these process actions would occur, and the time (starting at 0) when they would occur.

The set of processes in simulation 2 does not deadlock, but obviously other sequences can (see the samples for several such sequences). To detect deadlock, your program must build a resource graph (with nodes for processes and resources, and directed edges indicating resource ownership and pending resource requests), and test the graph for the existence of a cycle after each resource allocation request is made by a process. This test must be performed after each successful or unsuccessful allocation.

**The Instructor’s Solution**

The instructor’s solution (in executable form) is available in the file `/home/stanw/csci4500/prog2`. By default, the program will read its input data from the standard input. Alternately, the name of an input file may be specified on the command line. The instructor’s solution also provides a trace facility that may help you understand the sequence of events in a simulation. This trace is turned on if the \(-v\) option is specified on the command line. Thus the program invocation for the instructor’s solution has this syntax:
The trace output contains two types of information. Each time an action is attempted, the trace output will display the simulation time, the process identification, and the action involved. As you can see from the sample output (below), the computation time parameter will continually be reduced as the process makes progress toward completion.

The second type of information – shown in parentheses in the sample output – indicates when a resource is allocated, when an allocation was unsuccessful, when a resource was deallocated (released), when a process was unblocked (awakened) because a resource on which it was waiting was deallocated, and finally when a process terminated. This should all be obvious from examination of the sample output.

The Input Data

The input data will consist of a sequence of simulation test cases identified by sequential numbers starting with 1. The first line of input for each test case will contain two integers that specify the number of processes (1 ≤ np ≤ 50) and the number of resources (1 ≤ nr ≤ 50). The remaining np lines in the test case specify the actions for each of the np processes, with line i + 1 of the test case input containing the action specifications for process i.

Each action specification line contains an integer na (1 ≤ na ≤ 50) that specifies the number of actions for the process and then na action specifications, each consisting of an action type (L, U, or C) followed immediately by an integer. The last test case will be followed by a line containing two integer zeroes. Sample input and output appears below, and illustrates the input format.

The Assignment

Write and test a program (in C, C++, Java, or Python) to carry out the simulation of multiple processes executing the sequences of resource requests and computation specified by the input data, checking after each resource allocation request for the existence of deadlock by attempting to find a cycle in the resource graph. If no deadlocks occur during execution, then for each process indicate the total run time and the time when the process completed. If a deadlock does occur, indicate that fact, the time when the deadlock was detected, and the identification of the processes and resources involved in the circular wait. The samples shown below illustrate the desired output format. No extraneous output (e.g. debugging information) should appear in your output, and tracing information should not appear in the output from your solution.

The program must read from the standard input and write to the standard output. The solution must be submitted by Tuesday, April 10, 2018 by 11:59 PM. To submit the source code for your solution (which should be named prog2 with a suitable suffix), place it and a suitable makefile in a directory named csci4500-181-prog2 located just below your home directory on loki. (If your submission requires “special consideration” you may add a README file to the submission directory.) You will need to create this directory yourself. Do not put other files in this directory, and do not alter the contents of the directory after the due date! In particular, there should be no object or executable files, or input or output data files in the directory.

You may use any cycle detection algorithm you wish to detect deadlock. A reasonably explicit description of a simple cycle detection algorithm that is appropriate for this problem is provided on Loki in the file named cycle.txt in the csci4500 directory.

All processes will have at least one action item.

No invalid execution sequences will appear in the input. In particular, no process will attempt to release a resource it does not already possess, and no process will attempt to obtain a resource it already possesses. (There is only one unit of each resource available.)
A “template” solution in C has been (or will soon be) provided on loki in the file /home/stanw/csci4500/prog2_temp.c. This is not a complete solution, but the parts of the template that require additional development have been appropriately marked.

You may work in groups of no more than three to complete this assignment. If you do work in a group, make certain that you (a) have a single submission directory for one of the members in the group (and no submission directories for the other member(s)), and (b) include the names of the members of the group in a comment near the top of the source code. Thus if (fictitious persons) Jacob, Moshe, and Ethel worked together on a solution, then only one of them should have a submission directory, and it should contain source code that includes their names in a comment near the top of the source code file.

Note that you are not required to work in a group. But even if you work by yourself, it is still appropriate to include your name as the author in the source code.

Sample Input and Output
The sample input shown below is in the file /home/stanw/csci4500/prog2.input on loki. It includes five test cases. The output was produced from the instructor’s solution for this assignment. Although the samples are representative of the input that will be used to test your solution, you should certainly create additional test cases yourself.

Sample Input
1 2
5  L1  L2  C10  U1  U2
3 3
5  L1  L2  C2  U1  U2
5  L1  L2  C2  U1  U2
4  L3  C5  U3  C2
2 2
5  L1  L2  C2  U1  U2
5  L2  L1  C2  U2  U1
3 3
6  L1  C3  L2  C3  U1  U2
6  L2  C3  L3  C3  U2  U3
6  L3  C3  L1  C3  U3  U1
7 6
7  L1  C5  L2  C5  U1  U2  C3
4  C3  L3  C5  U3
5  C3  L2  C2  U2  C3
6  L4  C6  L3  U4  C2  U3
6  L3  C5  L5  U3  U5  C2
7  L6  C5  L2  C2  U2  U6  C3
7  L5  C2  L4  C5  U5  C2  U4
0 0

Expected Output for the Sample Input (without trace)
Simulation 1
All processes successfully terminated.
Process 1: run time = 14, ended at 14

Simulation 2
All processes successfully terminated.
Process 1: run time = 6, ended at 12
Process 2: run time = 6, ended at 21
Process 3: run time = 9, ended at 19

Simulation 3
Deadlock detected at time 2 involving...
  Processes 1, 2
  Resources 2, 1

Simulation 4
Deadlock detected at time 12 involving...
  Processes 1, 2, 3
  Resources 2, 3, 1

Simulation 5
Deadlock detected at time 40 involving...
  Processes 4, 5, 7
  Resources 3, 5, 4

Expected Output for the Sample Input (with trace enabled)

Simulation 1
0: process 1: L1
   (resource 1 allocated to process 1)
1: process 1: L2
   (resource 2 allocated to process 1)
2: process 1: C10
3: process 1: C9
4: process 1: C8
5: process 1: C7
6: process 1: C6
7: process 1: C5
8: process 1: C4
9: process 1: C3
10: process 1: C2
11: process 1: C1
12: process 1: U1
   (resource 1 released)
13: process 1: U2
   (resource 2 released)
   (process 1 terminated)
14: process 3: U3
   (resource 3 released)
   (process 3 terminated)
15: process 2: L1
   (resource 1 allocated to process 2)
16: process 2: L2
   (resource 2 allocated to process 2)
17: process 2: C2
18: process 2: C1
19: process 2: U1
   (resource 1 released)
20: process 2: U2
   (resource 2 released)
   (process 2 terminated)

All processes successfully terminated.
Process 1: run time = 14, ended at 14
Process 2: run time = 6, ended at 12
Process 3: run time = 9, ended at 19

Simulation 2
0: process 1: L1
   (resource 1 allocated to process 1)
1: process 1: L1
   (resource 1 unavailable)
1: process 3: L3
   (resource 3 allocated to process 3)
2: process 1: L2
   (resource 2 allocated to process 1)
3: process 3: C5
4: process 1: C2
5: process 3: C4
6: process 1: C1
7: process 3: C3
8: process 1: U1
   (resource 1 released)
9: process 3: C2
10: process 2: L1
   (resource 1 allocated to process 2)
11: process 1: U2
   (resource 2 released)
   (process 1 terminated)
12: process 3: C1
13: process 2: L2
   (resource 2 allocated to process 2)
14: process 3: U3
   (resource 3 released)
   (process 3 terminated)
15: process 2: C2
16: process 3: C2
17: process 2: C1
18: process 3: C1
19: process 2: U1
   (resource 1 released)
20: process 2: U2
   (resource 2 released)
   (process 2 terminated)

All processes successfully terminated.
Process 1: run time = 6, ended at 12
Process 2: run time = 6, ended at 21
Process 3: run time = 9, ended at 19

Simulation 3
0: process 1: L1
   (resource 1 allocated to process 1)
1: process 2: L2
   (resource 2 allocated to process 2)
2: process 1: L2
   (resource 2 allocated to process 2)
   (resource 2 unavailable)
2: process 2: L1
   (resource 1 unavailable)
Deadlock detected at time 2 involving...
Processes 1, 2
Resources 2, 1

Simulation 4
0: process 1: L1  
  (resource 1 allocated to process 1)
1: process 2: L2  
  (resource 2 allocated to process 2)
2: process 3: L3  
  (resource 3 allocated to process 3)
3: process 1: C3
4: process 2: C3
5: process 3: C3
6: process 1: C2
7: process 2: C2
8: process 3: C2
9: process 1: C1
10: process 2: C1
11: process 3: C1
12: process 1: L2  
  (resource 2 unavailable)
12: process 2: L3  
  (resource 3 unavailable)
12: process 3: L1  
  (resource 1 unavailable)
Deadlock detected at time 12 involving...
  Processes 1, 2, 3
  Resources 2, 3, 1

Simulation 5
0: process 1: L1  
  (resource 1 allocated to process 1)
1: process 2: C3
2: process 3: C3
3: process 4: L4  
  (resource 4 allocated to process 4)
4: process 5: L3  
  (resource 3 allocated to process 5)
5: process 6: L6  
  (resource 6 allocated to process 6)
6: process 7: L5  
  (resource 5 allocated to process 7)
7: process 1: C5
8: process 2: C2
9: process 3: C2
10: process 4: C6
11: process 5: C5
12: process 6: C5
13: process 7: C2
14: process 1: C4
15: process 2: C1
16: process 3: C1
17: process 4: C5
18: process 5: C4
19: process 6: C4
20: process 7: C1
21: process 1: C3
22: process 2: L3  
  (resource 3 unavailable)
22: process 3: L2  
  (resource 2 unavailable)
23: process 4: C4
24: process 5: C3
25: process 6: C3

26: process 7: L4  
  (resource 4 unavailable)
26: process 1: C2
27: process 3: C2
28: process 4: C3
29: process 5: C2
30: process 6: C2
31: process 1: C1
32: process 3: C1
33: process 4: C2
34: process 5: C1
35: process 6: C1
36: process 1: L2  
  (resource 2 unavailable)
36: process 3: U2  
  (resource 2 released)
  (process 1 unblocked)
37: process 4: C1
38: process 5: L5  
  (resource 5 unavailable)
38: process 6: L2  
  (resource 2 allocated to process 6)
39: process 1: L2  
  (resource 2 unavailable)
39: process 3: C3
40: process 4: L3  
  (resource 3 unavailable)
  (resource 3 unavailable)
Deadlock detected at time 40 involving...
  Processes 4, 5, 7
  Resources 3, 5, 4