CSCI 4404/5401 - Assignment-3
Due: 15th May, 2019 (Wednesday)

Please carefully read and adhere to **all** the below guidelines:

1. This Assignment is for **100 points** for both **CSCI 4401** and **CSCI 5401** students. It is highly recommended that students work on all the questions in the assignment as this will increase your chances of getting the full 100 points.
2. The Assignment scores will be reduced to 10 points for the final grade. For example, if a student scores 95 points in this assignment, the student will get 9.5 points credit towards the course grade.
3. Please don't collaborate when working on this assignment. The work submitted should be your own.
4. You are allowed to use C, C++, Java or Python for this assignment. **It will not be necessary to use any external libraries for this assignment**. Please don't use any.
5. Your uploaded code should be **compilable**. Make sure to include all the source code files.
6. **Make sure to include the output for the two test input files given** (input1.txt and input2.txt)
7. **Include a PDF file** to show any special compilation instructions that are needed to compile and run your code. Also, include instructions on how to run your program in the PDF file (i.e. how to feed it the name of the input file).

There are a variety of approaches to deal with resource deadlocks. For this assignment you'll investigate deadlock detection. One approach to deadlock detection that we discussed in class is to model resource allocations and requests with a Resource Allocation Graph (RAG). Deadlock detection is then accomplished by finding a cycle in the graph.

Your task is to construct the RAG and detect any cycles that arise. Your program will track resource requests, allocations and releases. The ordering of resource requests and releases will be provided as input to your program in the format given below. When a request is made for a resource, it will be allocated if it is available. Otherwise, the requesting process will be blocked until the resource is released by the process that holds it.

Your program should check for the presence of deadlock (a cycle) after every resource request or allocation. In a real system, this check would only be done periodically because it is too expensive to check after every request or allocation. However, your program is required to indicate the deadlock as soon as it occurs. Note that it is not necessary to check for deadlock after a resource release, because if deadlock didn't exist before the release then it won't exist afterward.

When your program detects a deadlock, it will report the deadlock along with the resources and processes involved in the cycle and then terminate. In a real system employing deadlock detection some attempt would be made to resolve the deadlock. In this case, however, we are only interested in detecting it. If your program performs all of the resource allocations and releases without encountering a deadlock, then it will report that no deadlock was encountered and terminate.

Input:

Input to your program will consist of lines like the following, read from a file:

```
1    W    1
2    W    2
3    W    6
2    W    6
4    W    3
5    W    1
1    R    1
3    W    3
5    R    1
4    W    2
1    W    4
1    W    5
1    R    4
```

Each line of input contains an integer followed by 'W' or 'R' and then another integer. The first integer identifies the process, the second integer identifies the resource, and 'W' indicates "wants" or 'R' indicates "releases". So the first line indicates that process 1 wants resource 1. The last line indicates that process 1 releases resource 4. All process and resource identifiers are unique, so the process 1 that is referred to five times in the input is the same process. Also, **each resource has only one instance**. Hence the resource can be allocated to only one process at a time. If a process wants

a resource that is already allocated then it will have to wait until the resource is released by whichever process is holding it. When a resource is released, it should be allocated to the process that has been waiting for it the longest. If there are no processes waiting for it then the resource is now free. Note that all input will be consistent. That is, I will never specify in the input file that process 7 Releases resource 3 if process 7 didn't hold resource 3.

Output:

Your program will write information to standard output that tracks the requests, allocations, and releases. The following format is required:

Process 1 wants resource 1 – Resource 1 is allocated to process 1.
Process 2 wants resource 2 – Resource 2 is allocated to process 2.
Process 3 wants resource 6 – Resource 6 is allocated to process 3.
Process 2 wants resource 6 – Process 2 must wait.
Process 4 wants resource 3 – Resource 3 is allocated to process 4.
Process 5 wants resource 1 – Process 5 must wait.
Process 1 releases resource 1 – Resource 1 is allocated to process 5.
Process 3 wants resource 3 – Process 3 must wait.
Process 5 releases resource 1 – Resource 1 is now free.
Process 4 wants resource 2 – Process 4 must wait.
DEADLOCK DETECTED: Processes 2, 3, 4, and Resources 2, 3, 6, are found in a cycle.

If your program processes all of the input and finds no deadlock then it should display the line:
"**EXECUTION COMPLETED: No deadlock encountered.**" as the last line of the output.

Note:

You should not assume any bound on the number of processes or resources. The integer identifier can range freely and you should expect larger numbers to test your program. That is, don't read the identifier as if it's a character, read it in as an integer (or a string if you want to be more general and allow names for your processes and resources) so your program handles multiple digit identifiers.

Some sample input files are given to you (input1.txt, input2.txt). Make sure you include the outputs that your program generates for these inputs as well separately. Your program will be tested on more inputs as well.

Hint for a clean method to do this:
Use a self-defined simple directed graph structure (made of node objects that point to other node objects) to represent a RAG. Each node object can either be a resource or a process. Then, do a Depth First Search (DFS) with the help of recursion to traverse through all the nodes in a graph. During traversal, if you notice that any node repeats, you know that there is a cycle and its a deadlock.