COMP 322/L—Introduction to Operating Systems and System Architecture
Assignment #3—Banker's Algorithm

Objective:

To implement resource allocation and demonstrate deadlock avoidance using the Banker's algorithm.

Specification:

The program simulates resource allocation to requesting processes and demonstrates deadlock avoidance with the Banker's algorithm. A menu controls the operations, and each choice calls the appropriate procedure, where the choices are:

1) Enter parameters
2) Run the Banker's algorithm to determine a safe sequence
3) Quit program and free memory

Assignment:

- The program uses a claim graph consisting of processes, multi-unit resources, request edges, allocation edges, and claim edges to represent the state of allocated resources to processes.
- The graph can be represented by a set of arrays:
  - ***Resource vector:*** an *m*-element vector, where *m* is the number of resources and each entry **resource[*j*]** records the total number of units of resource *j*.
  - ***Available vector:*** an *m*-element vector, where *m* is the number of resources and each entry **available[*j*]** records the number of units of resource *j* that are available.
  - ***Max claims array:*** an *n*x*m*-element array, where *m* is the number of resources and *n* is the number of processes, and each entry **maxclaim[*i*][*j*]** contains an integer that records the maximum number of units of resource *j* that process *i* may ever request.
  - ***Allocation array:*** an *n*x*m*-element array, where *m* is the number of resources and *n* is the number of processes, and each entry **allocation[*i*][*j*]** contains an integer that records the number of units of resource *j* that process *i* has actually been allocated.
  - ***Need array:*** an *n*x*m* array, where *m* is the number of resources and *n* is the number of processes, and each entry **need[*i*][*j*]** contains an integer that records the number of units of resource *j* that process *i* may need in the future.

What NOT to do (any violation will result in an automatic score of 0 on the assignment):

- Do NOT modify the choice values (1,2,3) or input characters and then try to convert them to integers--the test script used for grading your assignment will not work correctly.
- Do NOT turn in an alternate version of the assignment downloaded from the Internet (coursehero, chegg, reddit, github, etc.) or submitted from you or another student from a previous semester.
- Do NOT turn in your assignment coded in another programming language (C++, C#, Java).

What to turn in:

- The source code as a C file uploaded to Canvas by the deadline of 11:59pm PST (-20% per consecutive day for late submissions, up to the 4th day—note 1 minute late counts as a day late, 1 day and 1 minute late counts as 2 days late, etc.)
- Make sure your code compiles with the online C compiler before submitting: https://www.onlinegdb.com/online_c_compiler

Sample output

```
Banker's Algorithm
------------------
1) Enter parameters
2) Determine safe sequence
3) Quit program

Enter selection: 1
Enter number of processes: 5
Enter number of resources: 3
Enter number of units for resources (r0 to r2): 10 5 7
Enter max units process p0 will request from each resource (r0 to r2): 7 5 3
Enter max units process p1 will request from each resource (r0 to r2): 3 2 2
Enter max units process p2 will request from each resource (r0 to r2): 9 0 2
Enter max units process p3 will request from each resource (r0 to r2): 2 2 2
Enter max units process p4 will request from each resource (r0 to r2): 4 3 3
Enter number of units of each resource (r0 to r2) allocated to process p0: 0 1 0
Enter number of units of each resource (r0 to r2) allocated to process p1: 2 0 0
Enter number of units of each resource (r0 to r2) allocated to process p2: 3 0 2
Enter number of units of each resource (r0 to r2) allocated to process p3: 2 1 1
Enter number of units of each resource (r0 to r2) allocated to process p4: 0 0 2

        Units   Available
r0      10      3
r1      5       3
r2      7       2

        Maximum                 Current                 Potential
        r0      r1      r2      r0      r1      r2      r0      r1      r2
p0      7       5       3       0       1       0       7       4       3
p1      3       2       2       2       0       0       1       2       2
p2      9       0       2       3       0       2       6       0       0
p3      2       2       2       2       1       1       0       1       1
p4      4       3       3       0       0       2       4       3       1

Banker's Algorithm
------------------
1) Enter parameters
2) Determine safe sequence
3) Quit program

Enter selection: 2
Checking: < 7 4 3 > <= < 3 3 2 > :p0 could not be sequenced
Checking: < 1 2 2 > <= < 3 3 2 > :p1 safely sequenced
Checking: < 6 0 0 > <= < 5 3 2 > :p2 could not be sequenced
Checking: < 0 1 1 > <= < 5 3 2 > :p3 safely sequenced
Checking: < 4 3 1 > <= < 7 4 3 > :p4 safely sequenced
Checking: < 7 4 3 > <= < 7 4 5 > :p0 safely sequenced
Checking: < 6 0 0 > <= < 7 5 5 > :p2 safely sequenced

Banker's Algorithm
------------------
1) Enter parameters
2) Determine safe sequence
3) Quit program

Enter selection: 3
Quitting program...
```