COMP281 Assignment 2 Resit

- In the following, you will find 1 problem that constitute the Assignment 2 Resit. It will be also available on the *online judging (OJ)* system available at http://intranet.csc.liv.ac.uk/JudgeOnline/

- You need to write a valid C program that solves each of these problems – it must read the input, as specified in the problem description then print the solution to the given problem for that input.
    - Note that code is "correct" only if it **correctly implements a solution to the problem stated** in the assignment, not "if online judge accepts it".
    - That is, even if OJ accepts your code, it could be wrong. Read the problems carefully.

- Input is read from the standard input, in the same way that you read input from the keyboard as shown in lectures (e.g., using scanf). Output is also printed to the standard output, as you have seen (e.g., using printf).

- When you are satisfied that your programs work correctly, you must submit them through the departmental submission system, as described below.
    - Even if the program is not correct, still submit whatever you have! You can still earn points if certain parts of the code are done right.

- You must also include a brief report describing your solutions to the problems. This should be maximum one side of A4 paper (although there is no penalty for a longer report) and should give a description of how each of your solutions works. This should include describing the algorithm used to reach the solution, describing your use of any C language features (that were not discussed in lectures) and identifying any resources that you have used to help you solve the problems.

- This assignment is worth 30% of the total mark for COMP281.
    - All problems are weighted equally.
    - For each problem, you can earn a total of 20 points
        - 10 points for "Functionality and Correctness" awarded for programs that **correctly** solve the problem for all test cases.
        - 8 points for "Programming style, use of comments, indentation and identifiers" awarded depending on the style, comments and efficiency of the solution
        - 2 points for the quality and depth of the accompanying report
    - The final grade results from normalizing the earned points to a scale of 100.
    - See separate "COMP 281 Detailed Marking Guidelines" for more details.

Submission Instructions

- Name each of your c files according to the problem number; i.e., problem 10xx should be in a file 10xx.c (where 'xx' is replaced by the actual number).
- Place all of your C files, one per problem, and your report (in .pdf format) into a single zip file.
  - Please use the standard (pkzip) zip file format:
    https://en.wikipedia.org/wiki/Zip_%28file_format%29
    which is supported by winzip, winrar, etc. on windows/mac os X, and 'zip' on linux
  - test your zip file before submitting.
- Before you submit your solutions, please first test them using the online judge. **You are required to include the "Result" and the "RunID" in your C code as comments.** The OJ provides a RunID. **RunIDs are not problem IDs**.
  - Example:
    - the problem is 10xx
    - The solution has been accepted by the OJ, and the runID is 2033.
    - You add to your code: /* 2033 Accepted */ to 10xx.c
  - Result is one of the following: Accepted, Wrong Answer, Presentation Error, Time Limit Exceeded, Memory Limit Exceeded, Output Limit Exceeded, Runtime Error, Compile Error
- Submit this zip file using the departmental submission system at
  http://www.csc.liv.ac.uk/cgi-bin/submit.pl
  Only the file submitted through this link will be marked.
- The **deadline** for this assignment is **11th August 2017, 5pm.**
- Penalties for late submission apply in accordance with departmental policy as set out in the student handbook, which can be found at:
  http://intranet.csc.liv.ac.uk/student/ug-handbook.pdf

1. Problem 1076

Title: Matrix Multiplication of Sparse Matrices

Description:

In this exercise you have to write a program that can compute the matrix product of two **sparse** matrices that are represented as lists in the input format. (i.e., the matrices have 0 for all non-specified entries, for more information see the end of this assignment).

Make sure that you do not use more memory than needed to store the input.

Input:

The first number of the input, N, describes the number of entries of the first matrix A. Then the entries of A follow (see below), then a number, M, which indicates the number of entries of the second matrix B. Then the actual entries of B will follow.

The entries of the matrices A, B, are encoded, 1 per line, in the following format:

```
row_id column_id value
```

You can assume that entries are sorted first by row, then by column. The `values` themselves are integers, but may be negative (e.g., see the "-1" entry in the sample input)

Output:

A sparse matrix representation of the output matrix. Entries are sorted first by row, then by column.

Sample Input:

```
3
0 0 4
0 1 2
1 0 -1
2
1 0 10
1 1 -10
```

Sample Output:

```
0 0 20
0 1 -20
```

About Sparse Matrix Representations

A 'sparse' representation of a matrix, is one that only encodes the non-zero entries:
https://en.wikipedia.org/wiki/Sparse_matrix

The encoding that this assignment uses is the 'coordinate list' encoding, with the additional condition of ordering (i.e., "You can assume that entries are sorted first by row, then by column.").

The following bit of octave code illustrates what is going on:

```
$ octave
GNU Octave, version 3.8.2
Copyright (C) 2014 John W. Eaton and others.
This is free software; see the source code for copying conditions.
<...SNIP...>
Read http://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.

                                                                                  #the first matrix:
octave:1> A = [0,0,4; 0,1,2; 1,0,-1]
A =
   0   0   4
   0   1   2
   1   0  -1

                                                     #because octave does start indexing from '1':
octave:2> A=[ A(:,1)+1, A(:,2)+1. A(:,3) ]
A =
   1   1   4
   1   2   2
   2   1  -1

                                                                  #interpret A as a sparse matrix:
octave:3> As = spconvert(A)
As =
Compressed Column Sparse (rows = 2, cols = 2, nnz = 3 [75%])

  (1, 1) ->  4
  (2, 1) -> -1
  (1, 2) ->  2

                                                 #get the traditional 'dense' representation of A:
octave:4> Adense = full(As)
Adense =
   4   2
  -1   0

                                                                                 #the second matrix:
octave:5> B = [1,0,10; 1,1,-10]
B =
    1    0   10
    1    1  -10

                                                     #because octave does start indexing from '1':
octave:6> B=[ B(:,1)+1, B(:,2)+1. B(:,3) ]
B =
    2    1   10
    2    2  -10

                                                                    #interpret as a sparse matrix:
octave:7> Bs = spconvert(B)
Bs =

Compressed Column Sparse (rows = 2, cols = 2, nnz = 2 [50%])

  (2, 1) ->  10
  (2, 2) -> -10

                                                      #get the traditional 'dense' representation:
octave:8> Bdense = full(Bs)
Bdense =
    0    0
   10  -10

octave:9> C = Adense *Bdense
C =
   20  -20
    0    0

octave:10>
```