COMP281 Assignment 3 – C Programming

- **The following 4 problems will be available as Assignment 3 on the online judging system available at** http://intranet.csc.liv.ac.uk/JudgeOnline2/
- **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.**

- 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.

- You must also include a brief report describing your solutions to the problems. This should be up to 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), identifying any resources that you have used to help you solve the problems and describing any joint work you may have down with other students.

- This assignment is worth 40% of the total mark for COMP281

- 40% of the marks will be awarded for programs that correctly solve the problem for all test cases. All problems are weighted equally and each counts for 20% of this total.

- 50% of the marks will be awarded depending on the style, comments and efficiency of the solution. All problems are weighted equally and each counts for 20% of this total.

- 10% of the marks will be awarded for the quality and depth of the accompanying report

- See separate marking guidelines for more details.

Submission Instructions

- Name each of your c files with the problem number; i.e., 1046.c, 1048.c, 1049.c, and 1050.c. Place all four of your C files and your report (in .pdf format) into a single zip file.
- 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 March 18th, 4:00pm**

- Penalties for late submission apply in accordance with departmental policy as set out in the student handbook, which can be found at:
  http://www.csc.liv.ac.uk/student/ugpdfhandbook.pdf

Problem 1046

Title: Sum of Adjacent Numbers

Description

In the 3x3 grid below, the greatest sum of two adjacent numbers in any direction (up, down, left, right, or diagonally) is 22+23=45.

21 22 23

20 10 11

17 16 15

Input n and m. 1<=n<=100. 1<=m<=n. Then input a n x n grid. All integers in the grid are from 0 to 1000. Output the greatest sum of m adjacent numbers in any direction **(along straight line only)**.

You are required to use malloc to dynamically allocate just enough space for storing the grid.

Input

n, m, and then followed by a n x n grid

Output

The greatest sum of m adjacent numbers in any direction

Sample Input

```
3 2
21 22 23
20 00 11
17 06 15
```

Sample Output

```
45
```

Problem 1048

Title: Sort and Search for Dates

Description

Input n (1<=n<=10000) and then followed by n lines. Each line corresponds to a **valid** date, consisting of one string ("January", "February", ..., or "December"), one integer between 1 and 31, and one two digit integer representing the year (from 90 to 99, and then from 00 to 12). You do not have to worry about date validation. All dates in the input are valid dates.

Please use structures to store the dates. Please use malloc to dynamically allocate just enough space for n structures. You are asked to sort the dates chronologically using the built-in qsort function. Please output the

sorted list, one date per line, from most recent to oldest date. Please also use the built-in bsearch function to allow for a user query to check whether a specific date is in the list, and output either "Yes" or "No".

Input

n and then followed by n dates.

O***utput***

sorted list of dates, and whether a date input by the user (e.g. January 1 00) is in the list

Sample Input

```
10
January 1 01
January 1 00
February 28 99
July 17 12
September 10 12
July 1 00
June 30 90
August 25 06
May 27 08
October 1 03
January 1 00
```

Sample Output

```
September 10 12
July 17 12
May 27 08
August 25 06
October 1 03
January 1 01
July 1 00
January 1 00
February 28 99
June 30 90
Yes
```

Problem 1049

Title: Priority Queue

Description

A priority queue is an abstract data type, which is like a regular queue, but additionally, each element is associated with a "priority value". In this problem, the queue elements are integers from 10000 to 19999, the priority values are integers from 1 to 10 (10 corresponds to the highest priority).

The input consists of a list of operations, one operation per line. A sample input looks like the following:

Insert 10000 2

Insert 10000 2

Insert 10000 3

Insert 19444 9

Pop

Insert 10331 3

Pop

Pop

Pop

Pop

Pop

The operations are defined as follows:

- Insert x y: Add element x (10000<=x<=19999) to the queue with an associated priority value y (1<=y<=10). The queue allows duplicate elements. In the above example, we inserted 10000 three times. After these three insertions, there are two elements of value 10000 and priority 2 in the queue, and there is one element of value 10000 and priority 3 in the queue. You don't need to check the validity of the input. That is, if an input line says "Insert x y", then x is guaranteed to be between 10000 and 19999, and y is guaranteed to be between 1 and 10.
- Pop: Identify the element with the highest priority from the queue, output the element, and remove it from the queue. If the queue is empty, then output -1 (the queue remains empty). If there are multiple queue elements sharing the same highest priority value, then we remove and output the element that has the highest priority value and joined the queue the earliest.

For the above sample input, the sample output should be:

19444

10000

10331

10000

10000

-1

Explanation of the example: At the moment of the first "Pop", 19444 has the highest priority value 9. At the moment of the second "Pop", both 10331 and 10000 share the highest priority value 3. We remove and output 10000 because it joined the queue earlier. At the moment of the third "Pop", 10331 has the highest priority value 3. Finally, at the moment of the sixth "Pop", the queue is empty.

A priority queue can be implemented using a variety of methods. Your mark for this problem depends on the efficiency of your implementation.

Time efficiency: The time efficiency of your implementation is judged based on the speed of insert and pop when there are a huge number of elements in the queue.

Memory efficiency: The memory consumption of your queue should be dynamically adjusted based on the number of elements in it.

Input

A list of operations, one per line

Output

Corresponding output

Sample Input

```
Insert 10000 2
Insert 10000 2
Insert 10000 3
Insert 19444 9
Pop
Insert 10331 3
Pop
Pop
Pop
Pop
Pop
```

Sample Output

```
19444
10000
10331
10000
10000
-1
```

Problem 1050

Title: Flight Travel Planner

Description

You are working for a travel agency and are asked to write a computer program that can provide travel advice between cities based on a given map. The first line of the input of the program contains one integer n. n (2<=n<=10000) is the number of cities on the map. The second line of the input also contains an integer, m, (2<=m<=10000), which indicates the number of connections that will follow in the input below. Then, starting from the third line of the input, each line contains two integers from 1 to n, i.e. the connections between the cities. If a line says "a b", then it means that there is direct flight between city a and city b. There will be m of such connections. The last line also contains a pair of cities, say "c d", which will represent a query: "is there a connecting flight between c and d?"

Here is one sample input:

6

7

6 4

3 4

5 4

3 2

5 2

5 1

2 1

1 6

For this flight network, there exists at least one route between every pair of cities. For example, between city 5 and 6, there exists a route with 1 stop at city 4.

Given the input and the query, answer whether there is a route between cities c and d (note that this connection can be indirect). The output should be "Yes" or "No".

Let x be the total number of lines of the input. We know for sure that x is much smaller than n square. Your implementation should take this into consideration (for the purpose of saving memory).

For example, let us consider the case with n=10000 and x=10000 (so x is much less than n square). One easy way to represent a graph in memory is to use a n by n matrix. If the element in the i-th row and the j-th column equals 1, then that means node i and j are directly connected (0 means not directly connected). The memory consumption of the matrix representation is at least n square over 2 bits (each element only needs 1 bit; also, we only need to store half of the matrix due to symmetry). When n=10000, n square over 2 bits is just too much. You should find better ways to represent graphs in memory.

Input

Flight network

Output

Yes or No

Sample Input

```
6
7
6 4
3 4
5 4
3 2
5 2
5 1
2 1
1 6
```

Sample Output

```
Yes
```

Sample Input 2

```
5
3
4 3
3 2
5 1
1 4
```

Sample Output 2

```
No
```