SCHEDULING

Take your time to read this text very carefully. It contains some requirements you must absolutely follow. Any case of non-respect of requirements will have a negative effect on your grade.
Some other elements (clarification, complementary information) may be given you later.

PROGRAM TO DEVELOP

You program will perform the following actions:

1. Reading a constraint table from a .txt file, storing this information in memory and displaying the constraint table on screen.
2. Building a graph corresponding to that constraint table.
3. Checking that this graph has no circuits and that there are no negative arcs.
4. If all those properties are satisfied, compute the earliest date calendar, the latest date calendar, and the floats.

   When computing the latest date calendar, assume that the latest date of the end of project coincides with its earliest date. As you know, in order to compute the calendars, you must first do a topological sort of the graph, i.e. sort the vertices in the growing order of ranks. Therefore, you must find the ranks for all vertices using an algorithm of your choice among those you've seen in this course.

Organizing your work

You will form teams.

The number of students per team: 5 or 4 depending on the size of the group (no more than 8 teams per group).

Forming the teams: the list of teams must be put on Moodle by **a group delegate** (one Excel file per group) no later than **March 1**.

If you do not form teams by March 1, your teacher will split the group in teams on his or her own.

Programming language

You can choose between C, C++, Python, or Java

The language you chose must be sufficiently mastered by all members of the team, so that all of them could participate in the programming. During the project defense, your teacher can put any question to any team member.

Defenses

Presentation: 15 minutes + demonstration of your program/questions/answers/15 minutes.

The time allowed for a project defense is short. Your teacher will have a very dense schedule, which means it will not be possible to continue a defense after the end of the time slot.

Therefore, you'd better be ready at the scheduled moment, which implies:

- waiting at the door of the room and entering immediately when it's your turn;
- having prepared your computer, making sure that you have on it all the programs and files containing the constraint tables;
- having checked that your battery is charged;
- your computer must be on in the sleep mode;

- you have a replacement computer just in case, with the program and text files, and on which you have already checked that the program works as well as on the main one.

Every year there are students who think they'll never have a problem. Every year, there are those who do. The result: they get stressed and have less time for the defense. A pity.

Test constraint tables

The test tables (.txt files) will be given to you on **the 21 of March**. Those tables must be present on your computer at the presentation time.

DO not wait until we give you the test tables and test your program on any tables you can find or invent. Otherwise, it'll be too late. When developing your program, use all the constraint tables you've seen in lectures and in TDs, and as many others as you can. The more you test your program, the better you'll be certain it works as it should.

Sending in your work

All teams must put their work on Moodle no later than by **Sunday the 30th of March** (the depository will close at 00:00 of the 31th of March).

The content of what you must put on Moodle:

- Source code: Only the code files **that you typed yourselves** (**absolutely no file produced by the software during compilation or execution!**), well commented.
- All the .txt files of the test constraint tables (yes, even though it's us who'll give you the test tables), in the same directory as the code.
- A ppt or pdf of the presentation (this file may be provided during the presentation).
- The execution traces in the form of a .txt file. You will have to run your program on all the test tables and provide the corresponding execution traces.

A test constraint table that has not been tested, or for which the execution traces are not provided, will be considered as a case on which your program does not run correctly.

Constraint tables your program should be able to work with

Any constraint table of the following form (we take as an example the table C01 of the Exercise Sheet, and we've replaced the alphabetic task labels by numbers (A→1, B→2 etc.). On each line, the first number is the task number, the second is its duration, and the other numbers (if present) are the constraints (predecessors) :

1 9
2 2
3 3 2
4 5 1
5 2 1 4
6 2 5
7 2 4
8 4 4 5
9 5 4
10 1 2 3
11 2 1 5 6 7 8

The N tasks are numbered from 1 to N. The fictitious task α will be denoted as 0. The fictitious task ω will be numbered N+1.

Your program must be capable of importing any constraint table constructed as described above, including the case where the corresponding graph contains cycles or is not connected, and of transforming it into a graph in a matrix form (a value matrix).

Functions to program (to get at least 10 for your project you must succeed in programming the points from 1 to 4!)

The program

Set up a program that performs the following actions:

1. Reading a constraint table presented in a .txt file and storing it in memory
2. Display of the corresponding graph <u>in a matrix form</u> (value matrix). Warning: the display must be done from memory, not directly from reading the .txt file.
   The graph must contain the two fictitious tasks α and ω (labeled 0 and N+1 where N is the number of tasks).
3. Check the necessary properties of the graph such that it can serve as a scheduling graph
   - no cycle,
   - no negative edges.

   If those properties are satisfied, compute the calendars:

4. Compute the ranks for all vertices
5. Compute the earliest dates, the latest dates, and the floats.
   For the computation of the latest dates, assume that the latest date of the end of project coincides with its earliest date.
6. Compute the critical path(s) and display it or them

Your program must be capable to « loop » on the constraint tables you've prepared. It would be a very bad idea to stop the program and launch it again every time you want to use a different constraint table. If this is the case, it will results in points off.

The global structure of your program can be described by the following pseudo-code:

```
BEGIN
    WHILE the user wants to test a constraint table DO
        Choose the constraint table to work with
        Read it from a file and store it in memory
        Create the matrix of the graph corresponding to
        that constraint table, and display it
        Check the properties necessary for the graph to
        be a scheduling graph
        IF «yes» THEN
            Compute the ranks of all vertices and
            display them
```

```
                Compute the earliest dates calendar and
                the latest dates calendar and display
                them

                Compute the floats and display them

                Compute the critical path(s) and display
                it or them

            ENDIF

            ELSE ask the user if he wants to use another
            constraint table

        ENDWHILE

    END
```

It is evident that it is possible to incorporate detecting the absence/presence of a cycle and finding the ranks in one algorithm. However, you should display the ranks only in the absence of a cycle (s)

Execution Traces

Each of the steps should be accompanied by displaying the traces of what is being computed. Here's one example, which corresponds to the following constraint table:

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

We'll give here an example of display for the three first steps, and the display for the other steps should respect the same principle of legibility.

Step	Example of the trace (this is just an example : you can do what you want given that it allows one to understand what is being done fast and without any problem)

1	*Displaying the graph in form of triplets, edge by edge, for example:*
	`* Creating the scheduling graph:`
	`7 vertices`
	`9 edges`
	`0 -> 1 = 0`
	`0 -> 2 = 0`
	`1 -> 3 = 1`
	`1 -> 4 = 1`
	`2 -> 4 = 2`
	`2 -> 5 = 2`
	`3 -> 6 = 3`
	`4 -> 5 = 4`
	`5 -> 6 = 5`

<table>
<tr><td>2</td><td>

```
Representation of the graph in a value matrix form

Value Matrix
   0  1  2  3  4  5  6
0  *  0  0  *  *  *  *
1  *  *  *  1  1  *  *
2  *  *  *  *  2  2  *
3  *  *  *  *  *  *  3
4  *  *  *  *  *  4  *
5  *  *  *  *  *  *  5
6  *  *  *  *  *  *  *
```

(make sure that the matrix is properly displayed: the columns are well aligned, and the headings of both lines and columns are present)</td></tr>
<tr><td>3</td><td>There is one entry point, 0
There is one exit point, 6
Detecting cycles (For example, using the method of eliminating entry points) :

```
* Detecting a cycle
* Method of eliminating entry points
Entry points: 0
Eliminating entry points
Remaining vertices: 1 2 3 4 5 6
Entry points:  1 2
Eliminating entry points
Remaining vertices: 3 4 5 6
Entry points:  3 4
Eliminating entry points
Remaining vertices: 5 6
Entry points:  5
Eliminating entry points
Remaining vertices: 6
Entry points:  6
Eliminating entry points
Remaining vertices: None
-> There is no cycle
The values of outgoing edges for each vertex are the same
```

The edges 0->1 and 0->2 have a weight 0
There are no negative-weight edges
-> This is a scheduling graph</td></tr>
</table>