Practical 2 (due 2023-08-04 @ 09:00)

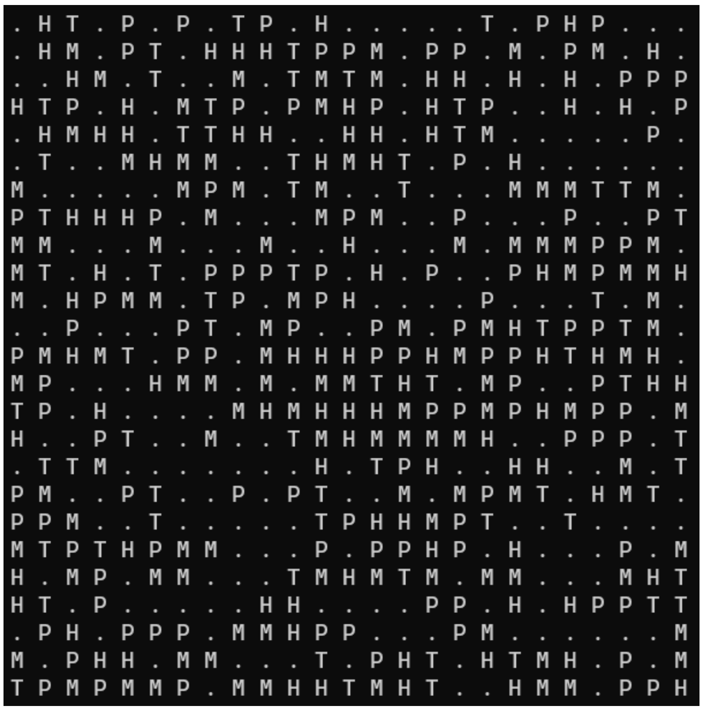
The purpose of this practical is for you to become familiar with the basics of C++ classes.

The Industrial Internet of Things (IIoT) consists of interconnected devices, machines and sensors to improve efficiency and productivity by enabling automation. IIoT is an extension on IoT to augment the capabilities of industrial sectors, such as energy, manufacturing, transportation, aviation, water management and smart cities. The theme for the practical assignments this year is analysing the data collected by IIoT sensors and using it to make meaningful decisions.

- Assume that an IIoT sensor collects the following telemetry data from an industrial plant. Create a data structure to represent an **`IIoTSensor`**. Use an appropriate C++ construct for creating data structures that only have data without functionality. The **`IIoTSensor`** must maintain the following information, including the range of values:
  - Sensor ID (*Integer* – `rowIndex * columnIndex`)
  - Temperature (*Double* – `[0, 50]`)
  - Humidity (*Integer* – `[0, 40]`)
  - Pressure (*Double* – `[0, 40]`)
  - Light Intensity (*Double* – `[0, 200000]`)

- Create a class to represent an **`IIoTMonitor`**, which manages a dynamic two-dimensional array of **IIoTSensor**s. The **`IIoTMonitor`** class must contain the following:
  - The number of rows (height) / columns (width) of the area covered
  - A two-dimensional dynamic array of sensors covering the area that is being monitored
  - A set of constructors for:
    - Creating a default **`IIoTMonitor`** (using associated class constants)
    - Creating **`IIoTMonitor`** where the dimensions of the area being monitored and a default value for all the sensors are specified
    - Creating a copy of an existing **`IIoTMonitor`**
  - A destructor
  - A `setSensorData` function that generates random sensor data for each of the **`IIoTSensor`** using the range of valid values. This function must be called in the constructor that does the initialisation
  - A `toString` function that returns a string (using `stringstream`) that representing sensors states. A malfunctioning or faulty sensor is defined as follows:
    - Temperature (*value <= 0 OR value > 40*)
    - Humidity (*value <= 0 OR value > 30*)
    - Pressure (*value <= 0 OR value > 30.2*)
    - Light Intensity (*value < 0 OR value > 100000*)
  - The `toString` function should return the following characters:
    - **M** if more than one sensor in the **IIoTSensor** is malfunctioning
    - **T** if only the temperature sensor is malfunctioning
    - **H** if only the humidity sensor is malfunctioning
    - **P** if only the pressure sensor is malfunctioning
    - **L** if only the light intensity sensor is malfunctioning
    - **.** if no sensor is malfunctioning
- Create a UML class diagram design as was demonstrated in class

Create a suitable `main` function which demonstrates the usage of the class. The `main` function must test the data structure by showing an **`IIoTMonitor`** environment similar to the one shown in Figure 1 in a `main.cpp` file (Hint: you will have to use the `toString` in the `main` file to display the environment).



```
. H T . P . P . T P . H . . . . . T . P H P . . .
. H M . P T . H H H T P P M . P P . M . P M . H .
. . H M . T . . M . T M T M . H H . H . H . P P P
H T P . H . M T P . P M H P . H T P . . H . H . P
. H M H H . T T H H . . H H . H T M . . . . . P .
. T . . M H M M . . T H M H T . P . H . . . . . .
M . . . . . M P M . T M . . T . . . M M M T T M .
P T H H H P . M . . . M P M . . P . . . P . . P T
M M . . . M . . . M . . H . . . M . M M M P P M .
M T . H . T . P P P T P . H . P . . P H M P M M H
M . H P M M . T P . M P H . . . . P . . . T . M .
. . P . . . P T . M P . . P M . P M H T P P T M .
P M H M T . P P . M H H H P P H M P P H T H M H .
M P . . . H M M . M . M M T H T . M P . . P T H H
T P . H . . . . M H M H H H M P P M P H M P P . M
H . . P T . . M . . T M H M M M M H . . P P P . T
. T T M . . . . . . . H . T P H . . H H . . M . T
P M . . P T . . P . P T . . M . M P M T . H M T .
P P M . . T . . . . . T P H H M P T . . T . . . .
M T P T H P M M . . . P . P P H P . H . . . P . M
H . M P . M M . . . T M H M T M . M M . . . M H T
H T . P . . . . . H H . . . . P P . H . H P P T T
. P H . P P P . M M H P P . . . P M . . . . . . M
M . P H H . M M . . . T . P H T . H T M H . P . M
T P M P M M P . M M H H T M H T . . H M M . P P H
```

Figure 1: A sample image representing a sample IIoT environment.

<u>Upload and submission</u>

- Create a PDF design document named **Design.pdf**
- When your program is working and you have created a design document, you must add your work to an archive file in the **zip** compression format. The name of the archive must be in the following format:

  `SURNAME_INITIALS_STUDENTNUMBER_SUBJECT_YEAR_P0.zip`

  e.g. for a student called Anne Student with student number 1234567

  `STUDENT_A_1234567_CSC01A1_2022_P0.zip`
- The archive must contain the following directories / folders:
  - **src** - containing the C++ source code needed to compile your program
  - **bin** – containing an executable generated from your source code
  - **doc** – containing your design document

Mark sheet		
	Design	10
	Class declaration / definition (in .h and .cpp file)	10
	Declaration of IIoTSensor structure	10
	Constructors	10
	Copy constructor	10
	Destructor	10
	setSensorData function	10
	toString function	10
	Utility member functions	10
	Demonstration of class functionality in a main function	10
	Total	/100