A. Introduction

The aim of this coursework is to develop the student's programming skills, particularly in the area of developing multithreaded applications. The students have to elaborate Windows console application *IAS0410ObjPlantLogger.exe* in C++ using the object-oriented programming concepts introduced in C++ versions up to 17. To get the highest mark, the students may complete his/her application with graphical user interface implemented in Qt framework.

B. Plant emulator

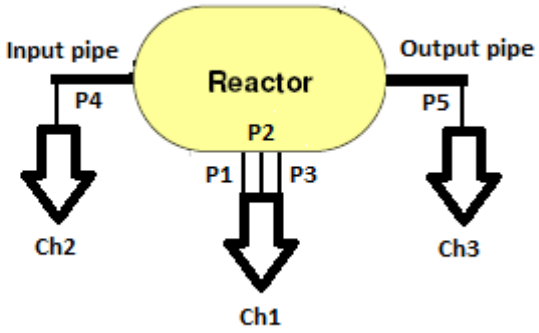
1. Basic concepts

Suppose we have a chemical plant consisting of several processing units like reactors, heaters, coolers, separators, tanks, etc. The processing units are connected with pipes.

The processing units and pipes are equipped with various measurement instruments recording the temperature, pressure, concentration of some substance in a solution, flow speed in pipes, etc. The results of measurements are sent to a computer which views them on screem and stores in a log file.

According to the terminology used in this coursework, we tell that the processing units and pipes have measurement points. Each measurement point has an instrument measuring a physical value (for example, the temperature in a reactor). The set of measurement points belonging to the same processing unit or pipe is the measurement channel.

Example:



The plant consists of reactor and two pipes. The measurement points P1, P2 and P3 are for measuring the temperature, pressure and liquid level in the reactor. Points P4 and P5 measure the liquid flow speed in pipes.

The measurement channel Ch1 incorporates measurement points P1, P2 and P3. Channels Ch2 and Ch3 contain only one point (P4 and P5 resepectively).

2. Emulator overview

In this coursework the chemical plants are replaced by emulating software implemented as a DLL. The Windows 64-bit (x64) *IAS0410PlantEmulator.dll* is able to emulate 10 different plants. The description of plants (i.e. the lists of channels,

points and measured values) is presented in JSON-formatted[1] file *IAS0410Plants.txt*. To get his/her plant number set by the instructor the student has to open page https://www.tud.ttu.ee/im/Viktor.Leppikson/IAS0410%20PracticalWork.html .

You may locate the emulator DLL into any folder. The *IAS0410Plants.txt* must in the same folder with *IAS0410ObjPlantLogger.exe.*

The emulator must be linked to the logger explicitly (see slides *Dynamic link libraries (6), (7)* and *(8)* from chapter *Concurrency*). It exports two functions:

```
LIBSPEC void SetIAS0410PlantEmulator(string, int) throw(exception);
LIBSPEC void RunIAS0410PlantEmulator(ControlData *) throw(exception);
```

As the extern "C" linking is applied, the name mangling is suppressed.

The *SetIAS0410PlantEmulator* function has two input parameters: the complete name and path of the *IAS0410Plants.txt* (you can get it processing the logger command line parameters) and the plant number. This function retrieves the plant description from JSON-formatted text and makes all the preparations necessary for running. In case of errors it throws C++ standard exception (see slides from chapter *Deeper into C++*).

Function *RunIAS0410PlantEmulator* launches a typical producer thread (see slide *Conditional variables (8), (9)* and *(10)* from chapter *Concurrency*). Consequently, the logger must launch a consumer thread. IMPORTANT: the producer thtread is detached.

The *ControlData* struct is defined as follows:

```
struct ControlData
{
	mutex mx;
	condition_variable cv;
	atomic<char> state;
	vector<unsigned char> *pBuf;
	promise<void> *pProm;

};
```

The logger uses attribute *state* to control the emulator:

- 'r' – The emulator launches the producer thread and keeps it running. The producer generates data and stores in vector **pBuf*. The length of vector is set by the emulator.
- 's' – The emulator forces the producer thread to exit.
- 'b' – The producer thread is running but temporarily does not generate data. To resume the normal working the logger must set the state back to 'r'.

The framework of producer thread is as follows:

```
void producer()
{
	// ..........................
	ControlData* pCd;
	while (true)
	{
		unique_lock<mutex> lock(pCd->mx);
		if (pCd->state == 's')
		{
			break;
```

[1] About JSON see http://www.json.org/

```
            }
            else if (pCd->state == 'b')
            {
                  // ..........................
            }
            // Create data, fill the buffer
            pCd->cv.notify_one();
            pCd->cv.wait(lock);
      }
      pCd->cv.notify_all();
      pCd->pProm->set_value(); // inform that the emulator has ended
}
```

3. Measurement results

The measurement results are delivered in packages. The emulator sends packages on random moments and the pauses between packets may be several seconds long.

The first package contains measurement results from all the points, i.e. all the channels and all the points are present. The contents of the following packages, however, is not preset – some of the points or even the complete channels may be omitted. Generally, the contents of packages except the first one is occasional.

Example (see also the figure above):

The first packet is complete:

Ch1: P1 = 49.6°C, P2 = 1.67atm, P3 = 98%; Ch2: P4 = 0.024m³/s; Ch3: P5 = 0.151m³/s [2].

The second packet contains three values:

Ch1: P1 = 55.2°C, P2 = 1.941atm; Ch3: P5 = 0.026m³/s.

The third packet contains only one value:

Ch2: P4 = 0.047m³/s.

The emulator does not send empty packages.

The measurement packages are formatted as follows:

1. The first four bytes of a packet are for storing the package length (i.e. the total number of bytes in package). It is a regular C/C++ integer.
2. The next four bytes present the number of channels included into the current package. It is also a regular C/C++ integer.
3. The following bytes are for the sequence of the channel packages.

Each channel package is formatted as follows:

[2] The values are created by generators of random numbers. Do not draw parallels between them and the real world.

1. The first four bytes present the number of measurement points included into the current channel package. It is a regular C/C++ integer.
2. The next bytes present the name of channel. It is a regular ASCII (not Unicode) C/C++ string including the terminating zero byte at the end.
3. The following bytes are for the sequence of the point packages.

Each point package is formatted as follows:

1. The starting bytes present the name of point. It is a regular ASCII (not Unicode) C/C++ string including the terminating zero byte at the end.
2. The following bytes present the measured value. It may be a four-byte integer or eight-byte double number.

Example: the first package described above includes the following bytes:
1. Bytes [0:3] – integer 83 as the total number of bytes in this package.
2. Bytes [4:7] – integer 3 as the number of channels in this package. After that without any delimiters the package presenting channel Ch1 begins.
3. Bytes [8:11] – integer 3 as the number of points in the package of channel Ch1
4. Bytes [12:15] – ASCII string "Ch1" with terminating zero byte. After that without any delimiters the sequence of packages presenting the points begins.
5. Bytes [16:18] – ASCII string "P1".
6. Bytes [19:26] – temperature measured at point P1 as double value.
7. Bytes [27:29] – ASCII string "P2".
8. Bytes [30:37] – pressure measured at point P2 as double value.
9. Bytes [38:40] – ASCII string "P3".
10. Bytes [41:44] – level measured at point P3 as integer value. This is also the end of Ch1 package. The Ch2 package follows.
11. Bytes [45:48] – integer 1 as the number of points in the package of channel Ch2
12. Bytes [49:52] – ASCII string "Ch2".
13. Bytes [53:55] – ASCII string "P4".
14. Bytes [56:63] – flow speed measured at point P4 as double value. This is also the end of Ch2 package. The Ch3 package follows.
15. Bytes [64:67] – integer 1 as the number of points in the package of channel Ch3
16. Bytes [68:71] – ASCII string "Ch3".
17. Bytes [72:74] – ASCII string "P5".
18. Bytes [75:82] – flow speed measured at point P5 as double value.

The second package described above includes the following bytes:
1. Bytes [0:3] – integer 57 as the total number of bytes in this package.
2. Bytes [4:7] – integer 2 as the number of channels in this package (Ch1 and Ch3 are present, Ch2 is omitted).
3. Bytes [8:11] – integer 2 as the number of points in the package of channel Ch1 (P1 and P2 are present, P3 is omitted).
4. Bytes [12:15] – ASCII string "Ch1".
5. Bytes [16:18] – ASCII string "P1".
6. Bytes [19:26] – temperature measured at point P1.
7. Bytes [27:29] – ASCII string "P2".
8. Bytes [30:37] – pressure measured at point P2.
9. Bytes [38:41] – integer 1 as the number of points in the package of channel Ch3
10. Bytes [42:45] – ASCII string "Ch3".

11. Bytes [46:48] – ASCII string "P5".
12. Bytes [49:56] – flow speed measured at point P5.

The third package described above includes the following bytes:
1. Bytes [0:3] – integer 27 as the total number of bytes in this package.
2. Bytes [4:7] – integer 1 as the number of channels in this package (only Ch2 is present).
3. Bytes [8:11] – integer 1 as the number of points in the package of channel Ch2.
4. Bytes [12:15] – ASCII string "Ch2".
5. Bytes [16:18] – ASCII string "P4".
6. Bytes [19:26] – flow speed measured at point P4.

C. Application *IAS0410ObjPlantLogger*

1. General requirements

The obligatory development environment is Microsoft Visual Studio (2019 or 2022).

To start with, tell the Visual Studio project wizard that you will develop a Windows Console Application.

The command line launching the application must have two arguments: the plant number and the log file name and path, for example:

IAS0410ObjPlantLogger 1 c:\testing\results.bin

The application must be able to complete the following tasks:
1. Attach and detach the emulator DLL.
2. Read data created by the emulator, add the timestamp and show it in Windows command prompt window.
3. Stop and restart the data stream from DLL into application.
4. Temporarily break and then resume the data generating and sending.
5. Store the data in log file.
6. Store the data inside application in a data structure.
7. Read data from log file into data structure.
8. Search data from data structure and show the results in Windows command prompt window.

The human ooperator controls the application with commands typed on the keyboard.

The application code must be object-oriented. All the functions except *main()* must be members of classes. All the attributes of classes must have private or protected access. Usage of the C/C++ *goto* statement is not allowed.

2. Behavior in abnormal situations

If something has failed (for example, the logger cannot attach the emulator, the producer thread itself terminates, data sent by emulator are not analysable etc.), the logger must show a message describing the situation and after that start to wait for the *"exit"* command.

It is not possible to get through the evaluation test with logger that crashes or hangs.

3. Data structure to be implemented in the logger

There are four versions:.

```
map<string, map<string, list<pair<variant<int, double>,
system_clock::time_point> > > > Data1;
/* It is a C++ map in which the channel names are the keys.The values are
inner C++ maps in which the keys are point names and the values are
lists.The members of lists are pairs in which member "first" is the
measument value and member "second" is the timestamp.*/

map<string, map<string, list<pair<variant<int, double>,
system_clock::time_point> > > * > Data2;
/* It is a C++ map in which the channel names are the keys.The values are
pointers to inner C++ maps in which the keys are point names and the
values are lists.The members of lists are pairs in which member "first" is
the measument value and member "second" is the timestamp.*/

map<string, map<string, list<pair<variant<int, double>,
system_clock::time_point> > * > > Data3;
/* It is a C++ map in which the channel names are the keys.The values are
inner C++ maps in which the keys are point names and the values are
pointers to lists.The members of lists are pairs in which member "first"
is the measument value and member "second" is the timestamp.*/

map<string, map<string, list<pair<variant<int, double>,
system_clock::time_point> > * > * > Data4;
/* It is a C++ map in which the channel names are the keys.The values are
pointers to inner C++ maps in which the keys are point names and the
values are pointers to lists.The members of lists are pairs in which
member "first" is the measument value and member "second" is the
timestamp.*/
```

To get his/her version set by the instructor the student has to open page https://www.tud.ttu.ee/im/Viktor.Leppikson/IAS0410%20PracticalWork.html.

4. Log file

The log file is a binary file specified in the logger command line. It simply contains packages created and sent by emulator. In other words, the logger writes the sequence of bytes it has received into log file. To each package the logger must append the timestamp:

```
system_clock::time_point t = system_clock::now();
```

When the logger starts to run, it must check is the log file empty or not. If there are data, the logger must read them and store into the data structure.

5. Keyboard commands controlling the logger

1. *"connect":* the logger attaches the DLL and calls method *SetIAS0410PlantEmulator()*.
2. *"disconnect":* the logger detaches the DLL. If the producer thread is running, this command should be ignored.
3. *"start":* the logger sets the state to 'r', clears the contents of buffer, launches the consumer thread and calls *RunIAS0410PlantEmulator()*. The emulator starts to generate data. The logger must add timestamp, show the measurement data on screen and store in data structure and in log file.

4. *"stop":* the logger sets the state to 's', thus terminating the producer and consumer threads. After that the operator may disconnect the DLL or restart the data generating.
5. *"break":* the logger sets the state to 'b'. The producer and consumer threads continue to run but the data is not generated.
6. *"resume":* the logger sets the state back 'r', normal working continues.
7. *"print"*: the logger shows the contents of data structure in Windows command prompt window. If the DLL is attached, this command should be ignored.
8. *"print channel_name"*: the logger retrieves from data structure all the data served by the specified channel and shows it in Windows command prompt window. If the DLL is attached, this command should be ignored. Example: *"print Ch2"*.
9. *"print channel_name point_name"*: the logger retrieves from data structure all the data served by the specified channel and specified mesurement point and shows it in Windows command prompt window. If the DLL is attached, this command should be ignored. Example: *"print Ch2 P4"*.
10. *"limits channel_name point_name"*: the logger finds from data structure the minimum and maximum values of data served by the specified channel and specified mesurement point and shows them in Windows command prompt window. If the DLL is attached, this command should be ignored. Example: *"limits Ch2 P4"*.
11. *"exit"*: must be applicable at any moment. If necessary terminates the threads, closes the log file, detaches the DLL, clears everything and quits.

If the command is senseless (for example *"start"* if the DLL is not attached or *"resume"* if the producer is not running"), the logger must print message "command ignored".

6. Output data formatting

Remark that the values created by emulator are random and may be absolutely unrealistic.

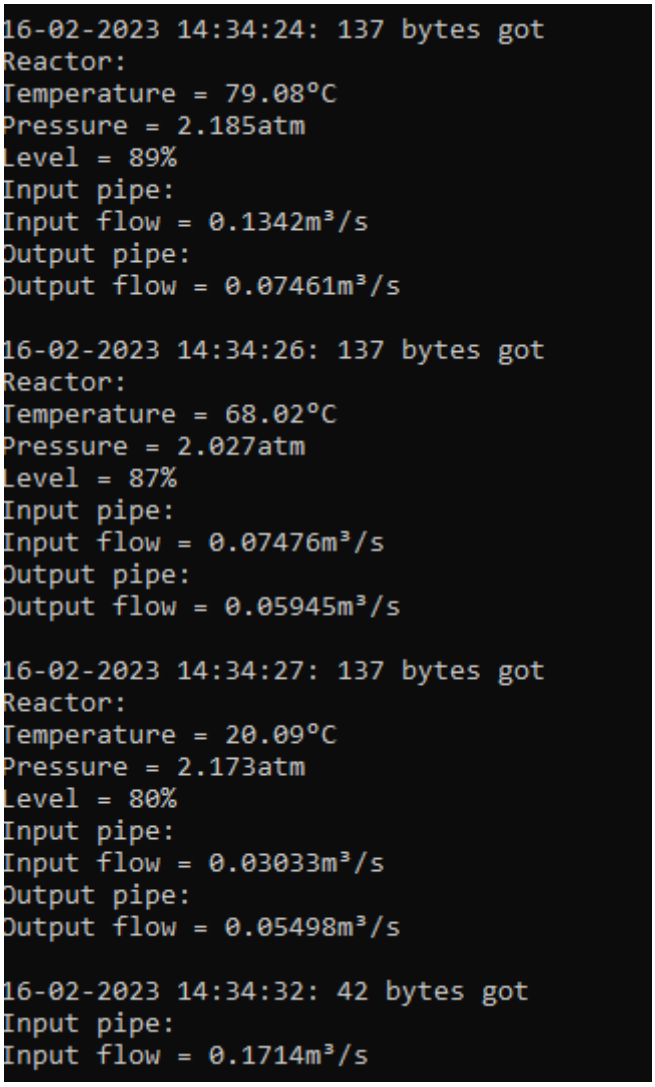
All the measured values must have units:

Value	Unit	printf formatting[3]
Temperature	°C	"%.1f"
Pressure	atm	"%.1f"
Concentration	%	"%d"
Level	%	"%d"
Kinematic viscosity	cSt	"%.2f"
Turbidity	NTU	"%.f"
Electrical conductivity	S/m	"%.2f"
Flow speed	m³/s	"%.3f"

[3] This column presents also the type (*int* or *double*) of values

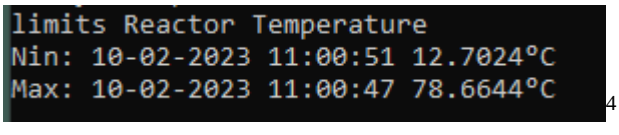
Quantity	kg	"%d"
Volume	L	"%d"
pH	-	"%.1f"

The printout in command promt window should be similar to the following example:



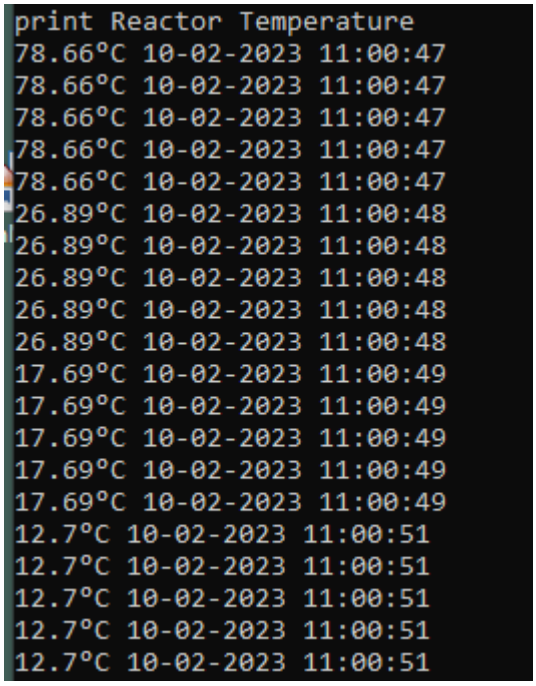
You are free to print debugging messages (for example, information about starting and terminating of threads, sending notifications, length of packages, etc.) into the command prompt window.

Here are examples of answers to inquiries:



[4]

[4] To find minimum and maximum use C++ stanard algorithm *minmax_element*



7. Hints

The command prompt window uses extended ASCII encoding in which the ˚ degree symbol is *F8* ja the ³ cube symbol is *FC*. The text files, however, use the UTF-8 encoding, in which ˚ degree symbol is *B0* ja the ³ cube symbol is *B3*. The code snippet

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
	fstream file;
	file.open("C:\\Temp\\Test.txt", fstream::out | fstream::trunc);
	double Temp = 13.5, Flow = 0.08;
	cout << "Temp = " << Temp << "°C" << endl;
	file << "Temp = " << Temp << "°C" << endl;
	cout << "Flow = " << Flow << "m³/s" << endl;
	file << "Flow = " << Flow << "m³/s" << endl;
	return 0;
}
```

gives you correct text in file but text in command prompt window may be corrupted. The problem is in Windows code pages[5]. To know the console default code page specified in your system registry write code snippet:

```
#include "Windows.h"
UINT WINAPI codepage = GetConsoleOutputCP();
cout << codepage << endl;
```

In codepage 1252 degree and cube symbols should be printed correctly:

```
#include <iostream>
#include <fstream>
#include "Windows.h"
using namespace std;
```

[5] See https://docs.microsoft.com/en-us/windows/console/console-code-pages and https://en.wikipedia.org/wiki/Windows_code_page

```
int main()
{
	UINT WINAPI codepage = GetConsoleOutputCP();
	SetConsoleOutputCP(1252);
	fstream file;
	file.open("C:\\Temp\\Test.txt", fstream::out | fstream::trunc);
	double Temp = 13.5, Flow = 0.08;
	cout << "Temp = " << Temp << "°C" << endl;
	cout << "Flow = " << Flow << "m³/s" << endl;
	file << "Temp = " << Temp << "°C" << endl;
	file << "Flow = " << Flow << "m³/sC" << endl;
	SetConsoleOutputCP(codepage);
	return 0;
}
```

8. Materials from the instructor

You may freely use (and also copy and paste) sections of code from the example programs presented and discussed in the lectures. Caution: the code of those examples may contain calls to functions that are nowadays considered to be unsecure and / or deprecated. It is your task to change them or suppress the Visual Studio warnings and compile error messages.

9. Test cases

The logger works perfectly if the following sequences of commands are performed correctly and according to the specification:

1. The log file must be empty. Launch the logger and apply the listed commands in the following order: connect → start → break → resume → stop → disconnect →print → print a channel data → print a point data → limits of a point → exit.
2. Use the same log file. Launch the logger and apply the listed commands in the following order: print → print a channel data → print a point data → limits of a point → connect → start → exit.
3. Use the same log file. Launch the logger and apply the listed commands in the following order: connect → disconnect →start →connect →start →disconnect →stop →start → break →exit.
4. Use the same log file. Launch the logger and apply the listed commands in the following order: connect → start → resume →stop → connect → print →start →start →stop → stop → exit.

Remark: the logger must ignore commands printed in red.

10. Evaluation and deadline

A student may earn up to 20 points:

- Application is fully in accordance with the specification and passes all the test cases – 20 points.
- Some test cases fail and / or the specification is not followed – 15 points.
- None of the test cases passes – 10 points.
- The code is not finished, compiling and / or linking is not possible – 5 points.

- The presented code is a partial or full plagiarism – 0 points.[6]
- Nothing is presented - 0 points.

The deadline is the end of session. However, the students may present the coursework before the deadline. The reception time is after each lecture in October, November and December.

The evaluation is provided only once and the number of earned points is final. There is no possibility to increase it later. To get the assessment the students must attend personally. Electronically (e-mail, git, etc.) sent coursework results are neither accepted nor reviewed.

Presenting the final release is not necessary. It is OK to demonstrate the work of your application in Visual Studio environment.

D. Application *IAS0410QtPlantLogger*

1. General requirements

The application must consist from two parts:

1. Graphical user interface (GUI) implemented in Qt framework.
2. DLL implemented in Visual Studio.

Here the DLL performs the same tasks as application *IAS0410ObjPlantLogger* specified above. The difference is that the commands controlling the logger are not typed on the keyboard but imported from GUI.

2. Hints

First implement application *IAS0410ObjPlantLogger* and make certain that it operates correctly. Follow all the requirements presented above.

As the second stage rework the application into DLL and implement the GUI[7].

The set of slides (see http://www.tud.ttu.ee/im/Viktor.Leppikson/) covering Qt programming contains also topics that are not in direct connection with problems you need to solve in our coursework. Therefore:

1. First install Qt and initialize the kits (slides *About Qt*, *Installation* and *First steps with QtCreator* #1 … #5).
2. Use QtCreator to implement (not just read but write the code, build the application and study what files you have got) some simple projects (slides *First steps with QtCreator* #6 … #11, examples *QtFirst* and *QtSecond* from *Example projects.zip*).
3. Elucidate for yourself the Qt mechanism of signals and slots (slides *Signals and slots*, examples *QtThird* and *QtFourth* from *Example projects.zip*).
4. Learn how to use layouts (slides *Layouts*) and try to build you GUI.

---

[6] However, copy and paste of code snippets from the material presented by intructor is allowed

[7] You may try to write the complete code in Qt, skipping the implementation of *IAS0410ObjPlantLogger*. However, do not forget that if you have finished with *IAS0410ObjPlantLogger*, you already have something to present for evaluation and failure with Qt does not lead to fatal consequencies. Turn attention that there will be no lectures about Qt and you have to learn the programming in Qt (only the basic knowledge is needed) on your own.

5. To understand how to deal with simple Qt threads study slides *QtThreads* #1 … #3 and example *QtSeventh* from *Example projects.zip*.
6. Read how to use third-.party DLLs in QT (slide *Third-party DLLs*)

3. Requirements on the graphical user interface

The graphical user interface must contain the following Qt widgets:

The **Open** *QPushButton* opens the *QFileDialog* dialog box thus allowing the user to select the log file. The button can be enabled only when the file is not selected.

The **Close** *QPushButton* closes the selected data file. It can be enabled only when the user has already opened the file and the emulator is not connected.

The **Connect** *QPushButton* is for sending the *Connect* keyboard command. It can be enabled only when the user has already opened the file but the emulator is not connected yet.

The **Disconnect** *QPushButton* is for sending the *Disconnect* keyboard command. It can be enabled only when the connection has been established but the emulator has not started yet.

The **Start** *QPushButton* is for sending the *Start* keyboard command. It can be enabled only when the connection has been established but the emulator has not started yet.

The **Break** *QPushButton* is for sending the *Break* keyboard command. It can be enabled only when the emulator is running.

The **Resume** *QPushButton* is for sending the *Resume* keyboard command. It can be enabled only when the data sending is temporarily broken off.

The **Stop** *QPushButton* is for sending the *Stop* keyboard command. It can be enabled only when the emulator is running.

The **Show** *QPushButton* is for sending the *Print*, *Print channel_name* and *Print channel_name point_name* keyboard commands. The channel name and the point name must be typed into **Channel** and **Point** *QLineEdit* boxes[8]. All the mentioned three widgets can be enabled only when the emulator is not connected. The results must be sent into **Logbook**.

The **Limits** *QPushButton* is for sending the *Limits Print channel_name point_name* keyboard command. The channel name and the point name must be typed into **Channel** and **Point** *QLineEdit* boxes.All the mentioned three widgets can be enabled only when the emulator is not connected. The results must be sent into **Logbook**.

The **Exit** *QPushButton* quits the application. It must be enabled all time.

The **Logbook** *QPlainTextEdit* is to inform the user about the download progress. After a package has arrived, it must immediately show its contents.

When an error has occurred (for example, opening the file failed), the application must inform the user displaying a message in logbook or in a *QMessageBox*.

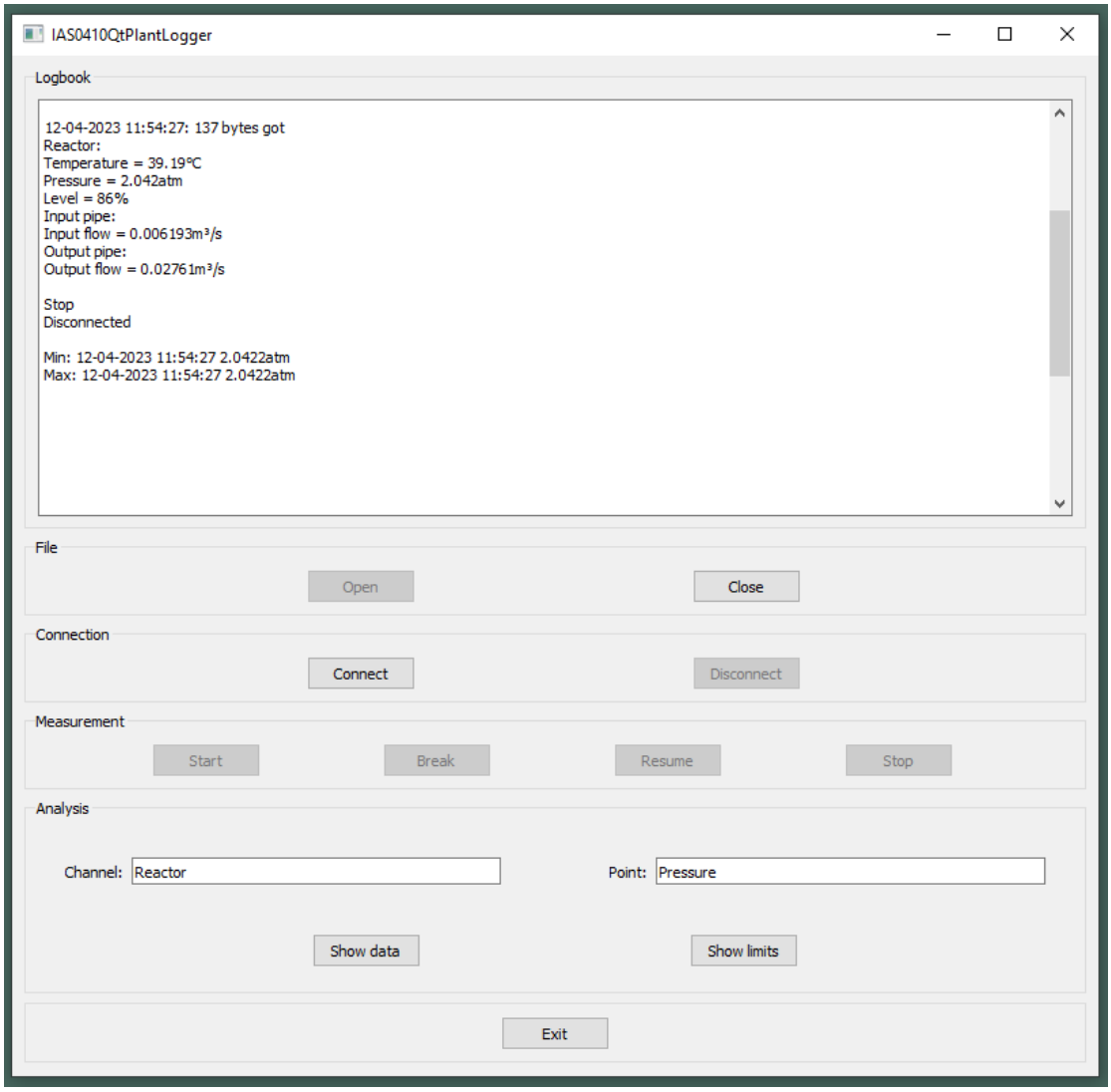
The **X** close button on the right up corner must operate as the **Exit** *QPushButton*.

---

[8] If the both *QlineEdit* boxes are empty, the *Print* command is supposed. If the **Point** box is empty, the *Print channel_name* command is supposed.

Enlarging and shrinking of the main window must not affect on the dimensions of buttons (i.e. the buttons must always have the same width and height). The *QPlainTextEdit* and *QLine Edit* dimensions should depend on the dimensions of the main window.

Example:



4. About cooperation between DLL and GUI

The GUI needs one additional thread. The GUI thread and the consumer thread from DLL comprise a typical producer / consumer pair sharing a buffer (now the consumer in DLL is the producer for GUI). The receiver stores the log data into file, converts into text and stores the text into buffer. The GUI thread (it must be an object of class derived from *QThread*) takes data from buffer and emits signal transporting the data to the main window. To synchronize their work Qt objects like *QMutex* and *QWaitCondition* are not applicable but you may replace them with C++ *std::mutex* and *std::condition_variable*. Use code from GUI to create synvhronization variables and copy the pointers to them into DLL.

The *IAS0410ObjPlantLogger* has a thread for communication with keyboard. In DLL it is not needed: instead of keyboard the commands now come from GUI. The Qt slots corresponding to signals from GUI buttons transfer the commands into DLL.

5. More hints

To convert log data to text use *stringstreams* like:

```
stringstream sout;
char *pBuf;
int bufLength;
double Temp = 13.5, Flow = 0.08;
sout << "Temp = " << Temp << "°C" << endl;
sout << " Flow = " << Flow << "m³/s" << endl;
strcpy_s(pBuf, bufLength, sout.str().c_str());
```

To view data in Qt window convert the C string from shared buffer *pBuf* to *QString*:

```
QString logData = QString::fromLocal8Bit(pBuf);
```

Usually the *IAS0410ObjPlantLogger* sends to *cout* also a lot of debugging information, for example the sequences of sent and read bytes in hex. You may redirect the output stream into a file. Read https://www.geeksforgeeks.org/io-redirection-c/.

6. Evaluation and deadline

A student may earn up to 30 points:

- Application is fully in accordance with the specification and passes all the test cases – 30 points.
- From esthetic point of view the GUI is unsatisfactory and / or the behavior of buttons is not in fully accordance with the specification – 25 points.
- The application is not able to work – 15 points.
- The code is not finished, compiling and / or linking is not possible – 10 points.
- The presented code is a partial or full plagiarism – 0 points.[9]
- Nothing is presented - 0 points.

The deadline is the end of session. However, the students may present the coursework before the deadline. The reception time is after each lecture in October, November and December.

The evaluation is provided only once and the number of earned points is final. There is no possibility to increase it later. To get the assessment the students must attend personally. Electronically (e-mail, git, etc.) sent coursework results are neither accepted nor reviewed.

Presenting the final release is not necessary. It is OK to demonstrate the work of your application in QtCreator environment.

E. Marks

The final mark (examination result) is computed from the sum of points:

- 26…30 points – "5".
- 20…25 points – "4".

[9] However, copy and paste of code snippets from the material presented by intructor is allowed

- 15…19 points – "3".
- 10…14 points – "2".
- 5…9 points – "1".
- 0…4 points – "0".

If a student has presented the *IAS0410ObjPlantLogger* and then comes with *IAS0410QtPlantLogger*, the points earned for *IAS0410ObjPlantLogger* are canceled.