

Homework - TCP server

Table of Contents

1. [Annotation](#)
2. [Task](#)
3. [Detailed specification](#)
 1. [Authentication](#)
 2. [Movement of robot to the target coordinates](#)
 3. [Secret message discovery](#)
 4. [Recharging](#)
4. [Error situations](#)
 1. [Error during authentication](#)
 2. [Syntax error](#)
 3. [Logic error](#)
 4. [Timeout](#)
5. [Special situations](#)
6. [Server optimization](#)
7. [Demo communication](#)
8. [Testing](#)
 1. [Tester](#)
 2. [Homework evaluation](#)
 3. [To download](#)
9. [Requirements to solution](#)
10. [Submission](#)
 1. [Upload to the archivation server](#)
 2. [Presentation in person](#)

Short instruction video for testing and quick start with homework: [homework-intro+testing_\(https://drive.google.com/file/d/1j-aggvlpSXdkOle9Anw9rd0mqRvn7r-q/view?usp=sharing\)](https://drive.google.com/file/d/1j-aggvlpSXdkOle9Anw9rd0mqRvn7r-q/view?usp=sharing).

Please find the demo source codes here: in Cpp: [demo-Cpp \(media/labs/homeworks/demo-C.zip\)](#) in Python: [demo-Python \(media/labs/homeworks/demo.zip\)](#)

Recording for implementation of sockets in C and python: [Sockets \(https://drive.google.com/file/d/1fzdwwJTp0EqYtwfxYPKdoR0rElqLvdHy/view?usp=sharing\)](https://drive.google.com/file/d/1fzdwwJTp0EqYtwfxYPKdoR0rElqLvdHy/view?usp=sharing).

Annotation

The task is to create multithreaded server for TCP/IP communication and implement communication protocol according to the specification. Note, the implementation of the client part is not a part of this task. See the section [Testing](#).



The server doesn't have to be implemented with threads, you can use processes or any other techniques, which allow the server to handle multiple clients in the same time.



Before implementation please read carefully [notes for submission](#)! By this you can avoid future complications.

Task

Create a server for automatic control of remote robots. Robots authenticate to the server and server directs them to origing of the coordinate system. For testing purposes each robot starts at the random coordinates and its target location is always located on the coordinate [0,0]. The robot should reach that target coordinate and pick the secret message, which can be found only on that spot. Server is able to navigate several robots at once and implements communication protocol without errors.

Detailed specification

Communication between server and robots is implemented via a pure textual protocol. Each command ends with a pair of special symbols "\a\b". Server must follow the communication protocol strictly, but should take into account imperfect robots firmware (see section Special situations).

Server messages:

Name	Message	Description
SERVER_CONFIRMATION	<16-bit number in decimal notation>\a\b	Message with confirmation code. Can contain maximally 5 digits and the termination sequence \a\b.
SERVER_MOVE	102 MOVE\a\b	Command to move one position forward
SERVER_TURN_LEFT	103 TURN LEFT\a\b	Command to turn left
SERVER_TURN_RIGHT	104 TURN RIGHT\a\b	Command to turn right
SERVER_PICK_UP	105 GET MESSAGE\a\b	Command to pick up the message
SERVER_LOGOUT	106 LOGOUT\a\b	Command to terminate the connection after successfull message discovery
SERVER_KEY_REQUEST	107 KEY REQUEST\a\b	Command to request Key ID for the communication
SERVER_OK	200 OK\a\b	Positive acknowledgement
SERVER_LOGIN_FAILED	300 LOGIN FAILED\a\b	Autentication failed

SERVER_SYNTAX_ERROR	301 SYNTAX ERROR\n\b	Incorrect syntax of the message
SERVER_LOGIC_ERROR	302 LOGIC ERROR\n\b	Message sent in wrong situation
SERVER_KEY_OUT_OF_RANGE_ERROR	303 KEY OUT OF RANGE\n\b	Key ID is not in the expected range

Client messages:

Name	Message	Description	Example	Maximal length
CLIENT_USERNAME	<user name>\n\b	Message with username. Username can be any sequence of characters except for the pair \n\b.	Oompa_Loompa\n\b	20
CLIENT_KEY_ID	<Key ID>\n\b	Message with Key ID. Can contain an integer with maximum of three ciphers.	2\n\b	5
CLIENT_CONFIRMATION	<16-bit number in decimal notation>\n\b	Message with confirmation code. Can contain maximally 5 digits and the termination sequence \n\b.	1009\n\b	7
CLIENT_OK	OK <x> <y>\n\b	Confirmation of performed movement, where x and y are the robot coordinates after execution of move command.	OK -3 -1\n\b	12
CLIENT_RECHARGING	RECHARGING\n\b	Robot starts charging and stops to respond to messages.		12
CLIENT_FULL_POWER	FULL POWER\n\b	Robot has recharged and accepts commands again.		12
CLIENT_MESSAGE	<text>\n\b	Text of discovered secret message. Can contain any characters except for the termination sequence \n\b.	Haf!\n\b	100

Time constants:

Name	Value [s]	Description
TIMEOUT	1	Server and client wait for the answer from the other side during this interval.
TIMEOUT_RECHARGING	5	Time interval, during which robot has to finish recharging.

Communication with robots could be divided into several phases:

Authentication

Server and client both know five pairs of authentication keys (these are not public and private keys):

Key ID	Server Key	Client Key
0	23019	32037
1	32037	29295
2	18789	13603
3	16443	29533
4	18189	21952

Each robot starts communication with sending its username (CLIENT_USERNAME message). Username can be any sequence of up to 18 characters not containing the termination sequence „\a\b“. In the next step the server asks the client for sending Key ID (SERVER_KEY_ID_REQUEST message), which is in fact an id of the server and client keys, which the client wants to use for authentication. The client answers by sending Key ID (CLIENT_KEY_ID message). After these steps the server knows the correct key pair and it can calculate "hash" code from the username of the client by the following formula:

Username: Meow!

ASCII representation: 77 101 111 119 33

Resulting hash: $((77 + 101 + 111 + 119 + 33) * 1000) \% 65536 = 47784$

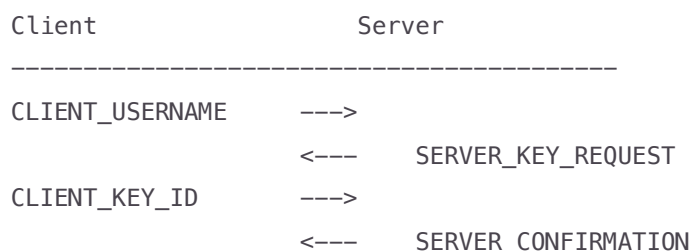
Resulting hash is 16-bit number in decimal notation. The server then adds a server key to the hash, so that if 16-bit capacity is exceeded, the value simply "wraps around" (due to modulo operation):

$(47784 + 54621) \% 65536 = 36869$

Resulting confirmation code of server is sent to client as text in SERVER_CONFIRM message. Client takes the received code and calculates hash back from it, then compares it with the expected hash value, which he has calculated from the username. If they are equal, client computes the confirmation code of client and sends it back to the server. Calculation of the client confirmation code is similar to the server one, only the client key is used:

$(47784 + 45328) \% 65536 = 27576$

Client confirmation key is sent to the server in CLIENT_CONFIRMATION message, server calculates hash back from it and compares it with the original hash of the username. If values are equal, server sends message SERVER_OK, otherwise answers with message SERVER_LOGIN_FAILED and terminates the connection. The whole sequence of steps is represented at the following picture:



```

CLIENT_CONFIRMATION ---->
    <----    SERVER_OK
            or
            SERVER_LOGIN_FAILED
    .
    .
    .

```

Server does not know usernames in advance. Robots can choose any name, but they have to know the list of client and server keys. The key pair ensures two-sided authentication and prevents the authentication process from being compromised by simple eavesdropping of communication.

Movement of robot to the target coordinates

Robot can move only straight (SERVER_MOVE), but is able to turn right (SERVER_TURN_RIGHT) or left (SERVER_TURN_LEFT). After each move command robot sends confirmation (CLIENT_OK), part of which is actual coordinates of robot. At the beginning of communication robot position is not known to server. Server must find out robot position and orientation (direction) only from robot answers. In order to prevent infinite wandering of robot in space, each robot has a limited number of movements (move forward). The number of turns is not limited. The number of moves should be sufficient for a reasonable robot transfer to the target. Following is a demonstration of communication. The server first moves the robot twice to detect its current state and then guides it towards the target coordinate [0,0].

```

Client          Server
-----
    .
    .
    .
    <----    SERVER_MOVE
            or
            SERVER_TURN_LEFT
            or
            SERVER_TURN_RIGHT
CLIENT_CONFIRM ---->
    <----    SERVER_MOVE
            or
            SERVER_TURN_LEFT
            or
            SERVER_TURN_RIGHT
CLIENT_CONFIRM ---->
    <----    SERVER_MOVE
            or
            SERVER_TURN_LEFT
            or
            SERVER_TURN_RIGHT
    .

```

•
•

This part of communication cannot be skipped, robot waits at least one of the movement commands - SERVER_MOVE, SERVER_TURN_LEFT or SERVER_TURN_RIGHT. There is several obstacles on the way to the target coordinate, which must be bypassed by the robots. The obstacle placement follows these rules:

- The obstacle spans only a single pair of coordinates.
- All neighbouring coordinates around the obstacle are always free, therefore the obstacle can be always bypassed.
- The obstacle is never placed on the target coordinates [0,0].
- If the robot crushes an obstacle (any obstacle) more than 20 times, it breaks down and ends the connection.

The obstacle is detected by the fact that robot doesn't change its position after SERVER_MOVE command (the CLIENT_OK message contains the same coordinate as it was in the previous step). If the robot doesn't move, the number of remaining moves is not decreased, but the number of remaining crashes is decreased by one.

Secret message discovery

After the robot reaches the target coordinate [0,0], it attempts to pick up the secret message (SERVER_PICK_UP). If robot receives command to pick up the secret message, but robot is not in the target coordinate [0,0], an autodestruction of robot is initiated and communication with server is abrupted. After the robot picks the secret message, it sends to the server CLIENT_MESSAGE with the secret. The server has to answer with SERVER_LOGOUT. (It is guaranteed, that secret message never matches the message CLIENT_RECHARGING, so if the recharge message is obtained by the server after the pick up command, it always means that robot started to charge.) After that, client and server close the connection. Demo of the secret message picking:

Client		Server

		•
		•
		•
	<---	SERVER_PICK_UP
CLIENT_MESSAGE	----	
	<---	SERVER_LOGOUT

Recharging

Each robot has a limited power source. If it starts to run out of battery, he notifies the server and then recharges itself from the solar panel. It does not respond to any messages during the charging. When it finishes, it informs the server and continues there, where it left before recharging. If the robot does not stop charging in the time interval TIMEOUT_RECHARGING, the server terminates the connection.

Client		Server

CLIENT_USERNAME	--->	
	<---	SERVER_CONFIRMATION
CLIENT_RECHARGING	--->	
...		
CLIENT_FULL_POWER	--->	
CLIENT_OK	--->	
	<---	SERVER_OK
		or
		SERVER_LOGIN_FAILED
		.
		.
		.

Another example:

Client		Server

		.
		.
		.
	<---	SERVER_MOVE
CLIENT_OK	--->	
CLIENT_RECHARGING	--->	
...		
CLIENT_FULL_POWER	--->	
	<---	SERVER_MOVE
CLIENT_OK	--->	
		.
		.
		.

Error situations

Some robots can have corrupted firmware and thus communicate wrongly. Server should detect misbehavior and react correctly.

Error during authentication

If the CLIENT_KEY_ID message contains the id outside of the expected range (which is a integer value between 0 and 4), the server sends the SERVER_KEY_OUT_OF_RANGE_ERROR and closes the connection. For simplification of

the implementation process the CLIENT_KEY_ID can contain even negative numbers (it doesn't make sense in the context of this protocol). If the CLIENT_KEY_ID message contains something else than integer values (for example letters), the server sends SERVER_SYNTAX_ERROR message and closes the connection.

If the CLIENT_CONFIRMATION message contains the incorrect value (even the negative value), the server sends SERVER_LOGIN_FAILED message and closes the connection. If the CLIENT_CONFIRMATION contains anything else than numeric value, the server reacts with SERVER_SYNTAX_ERROR message and closes the connection.

Syntax error

The server always reacts to the syntax error immediately after receiving the message in which it detected the error. The server sends the SERVER_SYNTAX_ERROR message to the robot and then terminates the connection as soon as possible. Syntactically incorrect messages:

- Incoming message is longer than number of characters defined for each message (including the termination sequence \ab). Message length is defined in client messages table.
- Incoming message syntax does not correspond to any of messages CLIENT_USERNAME, CLIENT_KEY_ID, CLIENT_CONFIRMATION, CLIENT_OK, CLIENT_RECHARGING and CLIENT_FULL_POWER.

Each incoming message is tested for the maximal size and only messages CLIENT_CONFIRMATION, CLIENT_OK, CLIENT_RECHARGING and CLIENT_FULL_POWER are tested for their content (messages CLIENT_USERNAME and CLIENT_MESSAGE can contain anything).

Logic error

Logic error happens only during the recharging - when robot sends information about charging (CLIENT_RECHARGING) and then sends anything other than CLIENT_FULL_POWER or it sends CLIENT_FULL_POWER without sending CLIENT_RECHARGING before. Server reacts to these situations with SERVER_LOGIC_ERROR message and immediate termination of connection.

Timeout

Protokol for communication with robots contains two timeout types:

- TIMEOUT - timeout for communication. If robot or server does not receive message from the other side for this time interval, they consider the connection to be lost and immediately terminate it.
- TIMEOUT_RECHARGING - timeout for robot charging. After the server receives message CLIENT_RECHARGING, robot should at latest till this interval send message CLIENT_FULL_POWER. If robot does not manage it, server has to immediately terminate the connection.

Special situations

During the communication through some complicated network infrastructure two situations can take place:

- Message can arrive divided into several parts, which are read from the socket one at a time. (This happens due to segmentation and possible delay of some segments on the way through the network.)
- Message, sent shortly after another one, may arrive almost simultaneously with it. They could be read together with one reading from the socket. (This happens, when the server does not manage to read the first message from the buffer before the second message arrives.)

Using a direct connection between the server and the robots, combined with powerful hardware, these situations cannot occur naturally, so they are artificially created by the tester. In some tests, both situations are combined.

Every properly implemented server should be able to cope with this situation. Firmware of robots counts on this fact and even exploits it. If there are situations in the protocol where messages from the robot have a predetermined order, they are sent in that order at once. This allows robots to reduce their power consumption and simplifies protocol implementation (from their point of view).

Server optimization

Server optimize the protokol so it does not wait for the end of the message, which is obviously bad. For example, only a part of the username message is sent for authentication. Server for example receives 22 characters of the username, but still does not receive the termination sequence `\a\b`. Since the maximum username message length is 20 characters, it is clear that the message received cannot be valid. Therefore, the server does not wait for the rest of the message, but sends a message `SERVER_SYNTAX_ERROR` and terminates the connection. In principle, server should react in the same way when receiving a secret message.

In the part of communication where robot is navigated to the target coordinates, server expects three possible messages: `CLIENT_OK`, `CLIENT_RECHARGING`, or `CLIENT_FULL_POWER`. If server reads a part of the incomplete message and this part is longer than the maximum length of these messages, it sends `SERVER_SYNTAX_ERROR` and terminates the connection. For the help with optimization, the maximum length for each message is listed in the table.

Demo communication

```
C: "0ompa Loompa\a\b"
S: "107 KEY REQUEST\a\b"
C: "0\a\b"
S: "64907\a\b"
C: "8389\a\b"
S: "200 OK\a\b"
S: "102 MOVE\a\b"
C: "OK 0 0\a\b"
S: "102 MOVE\a\b"
C: "OK -1 0\a\b"
S: "104 TURN RIGHT\a\b"
C: "OK -1 0\a\b"
S: "104 TURN RIGHT\a\b"
C: "OK -1 0\a\b"
```

```
S: "102 MOVE\a\b"
C: "OK 0 0\a\b"
S: "105 GET MESSAGE\a\b"
C: "Secret message.\a\b"
S: "106 LOGOUT\a\b"
```

Testing

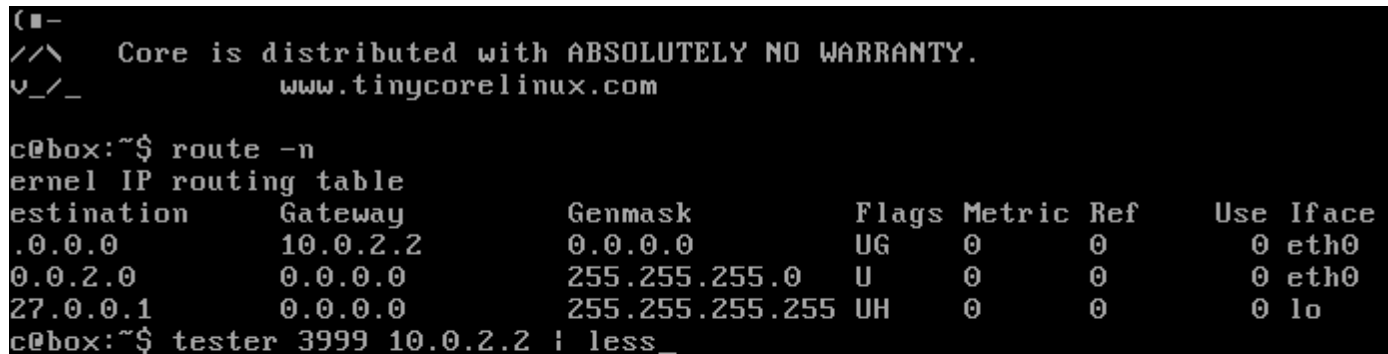
Image of OS Tiny Core Linux is prepared for your server testing. It contains tester of the homework. Image is compatible with VirtualBox application.

Tester

Download and unzip the image. Then run the image in VirtualBox. After starting and booting shell is immediately ready to use. Tester is run by command *tester*:

```
tester <port number> <remote address> [test number(s)]
```

The first parameter is the port number on which your server will listen. The following is a parameter with the server remote address. If your server is running on the same computer as VirtualBox, use the default gateway address. The procedure is shown in the following figure:



```
([ -
/\/ Core is distributed with ABSOLUTELY NO WARRANTY.
v/_/_ www.tinycorelinux.com

c@box:~$ route -n
Kernel IP routing table
Destination      Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0          10.0.2.2       0.0.0.0         UG    0      0      0 eth0
0.0.2.0          0.0.0.0        255.255.255.0   U     0      0      0 eth0
27.0.0.1         0.0.0.0        255.255.255.255 UH    0      0      0 lo
c@box:~$ tester 3999 10.0.2.2 | less_
```

The output is quite long, so it is good to redirect it to the *less* command, in which it is easier to navigate.

If no test number is entered, all tests are run sequentially. Tests can also be run individually. The following example runs tests 2, 3, and 8:

```
tester 3999 10.0.2.2 2 3 8 | less
```

Homework evaluation

The individual tests performed by the tester are divided into thematic sections. To successfully obtain the assessment, it is necessary to pass all the tests from the mandatory sections and it is possible to obtain 10 points for the assessment. The other parts are optional and a total of 10 more points can be obtained for them. Overview of individual sections:

- Ideal situation (mandatory)
- Authentication (mandatory)
- Segmentation and merging (mandatory)
- Timeout (mandatory)
- Robot movement (mandatory)
- Syntax errors
- Optimalization
- Charging
- Multi-threading
- Random test
- Final test (three random test instances run in parallel)

Upon completion of tests, the tester will print a report on the results.

Possible problems in OS Windows and Mac

In some Windows or Mac OS installations, there is a problem with standard configuration of virtual machine. If the tester in virtual machine cannot connect to the tested server in the host operating system, follow these steps:

- Additional step for Mac OS only, as Host-only Adapters in VirtualBox on do not work right out of the box: creating a Host-only Network in VirtualBox (File → Host Network Manager: "Create" button).
- When virtual machine with tester is off, change its network adapter settings from NAT to Host-only network (Host-only Adapter). In MacOS, select "vboxnet0" under the "name" drop down list.
- The network interface belonging to VirtualBox should appear in the host OS. This can be found from the command line with the *ipconfig* command. The IP address of this interface is likely to be 192.168.56.1/24.
- Now you need to manually set the IP address of eth0 network interface in the virtual machine with tester:

```
sudo ifconfig eth0 192.168.56.2 netmask 255.255.255.0
```

- Now you can start the tester but as the destination address enter the IP address of the network interface in the host OS:

```
tester 3999 192.168.56.1
```

- Do not forget to use that address in your server.

To download

VirtualBox: <https://www.virtualbox.org/wiki/Downloads>

Tester: [All version of the tester \(https://drive.google.com/drive/folders/1QzPyzZeLNWZhjtbaTGehyNu-zgHcInta?usp=sharing\)](https://drive.google.com/drive/folders/1QzPyzZeLNWZhjtbaTGehyNu-zgHcInta?usp=sharing)

The directory from the previous link contains directories with the various versions of the tester. Each directory has the version vX (where X is the number of the version). If you want to understand all secret messages, use the english version vX-EN. Always download the version with the greatest number X. Each directory consists of the following files:

- BI-PSI_tester_2023_vX-EN.ova - OS image with tester as a virtual machine for VirtualBox
- psi-tester-2023-vX-EN_x64.bz2 - Binary program of tester for linux (64-bit version)
- psi-tester-2023-vX-EN_x86.bz2 - Binary program of tester for linux
- psi-tester-2023-vX-EN_arm.bz2 - Binary program of tester for the ARM architecture

Users of Apple MacBook with the processor M1 have a slightly complicated testing process. The Apple company makes the creations of C++ applications much harder, if you don't work in Apple operating system. Noone in our team poses the target laptop, hence we cannot simply create and test the tester for that architecture. In order to run it, you need to install the VirtualBox from the link above (the preview version should work fine) and install any linux distribution for ARM architecture. The tester should work in that operating system just fine. The Debian distribution should do the trick:

[Debian install iso for arm64 \(https://cdimage.debian.org/debian-cd/current/arm64/iso-cd/debian-11.6.0-arm64-netinst.iso\)](https://cdimage.debian.org/debian-cd/current/arm64/iso-cd/debian-11.6.0-arm64-netinst.iso)

Requirements to solution

- The solution can be created in any programming language that implements the socket interface. The function or method of *receive* / *send* must be used for reading and writing to the socket. In Java, you can use streaming read and write directly on the socket.
- Only the solution that will pass all mandatory tests will be accepted. The tester shows the list of tests and which of them are mandatory. At the end the tester shows the final number of achieved points.
 - The mandatory tests can give you at most 10 points.

Submission

The task is only successfully submitted if the source code was uploaded to the submission server and the solution was personally presented on the lab. The date of submission is determined by the upload to the submission server, so it is possible to present the task after the deadline and without penalty.



DEADLINE for the upload without penalization is 30th of April, 2023!



After the deadline, the maximum number of points available will be reduced by two for each week started. It means that you need more than mandatory 10 points in order to compensate for penalization. For example, if the penalty is -2 points, you have to deliver the solution with the final score 12 points and better to reach the assesment.



The homework must be presented no later than the end of the assessment week!

On the other hand, early submission will be rewarded with points for activity. Each week in advance will be awarded with one point up to a maximum of 3 points. Schedule of possible bonus points:

- till 9th of April +3 points
- till 16th of April +2 points
- till 23th of April +1 points

Please note that **only 100% solution can get points for the activity**. Activity points will be awarded (or corrected) collectively for all solutions at the end of the semester, when the submission of homeworks is completed. The points will of course be calculated for the date of uploading to the archivation server before the presentation.

Here is the schedule of penalizations:

- from 1st of May -2 points
- from 8th of May -4 points
- from 15th of May -6 points



Having only mandatory part and submitting late imply failure to get assessment (10 points - penalization)!

Upload to the archivation server

A special server is used for submission. **Each student registers with it and will upload their ongoing versions of solution in their own interest, so that everyone has a traceable history in case of suspected plagiarism.** At the end of the semester, all submitted source codes will be tested for duplicities. If two or more codes match, submission history can help to identify the guilty one.



Source code is uploaded in one file and uncompressed (not archived/ziped).

The submission server does not check the code, it only compares it with the codes of other students and searches for matches. It is therefore possible to combine multiple source codes into one, even if such code could not be compiled

without modification.

Link to the submission server: [PSI bouda \(https://bouda.fit.cvut.cz\)](https://bouda.fit.cvut.cz).

Presentation in person

It is held at the labs. The student must demonstrate during the presentation that he/she understands the code and that the code works. The code that is presented must be the same as the one submitted to the submission server. The check is done in the following steps:

1. Student proves his identity.
2. Student shows the source code and runs the test to show that the code presented is the one being tested.
3. Student answers control questions to the source code.
4. Student uploads the source code to the submission server so that it is clear that he is uploading the code presented.

It is up to each student to ensure that all these steps run smoothly. The student calls the teacher for presentation only after he is ready for it. It is assumed that students present their solutions on their laptops, if you do not have one, consult with the teacher on how to present it.



Present the task as soon as possible after the final upload to the Bouda server. The longer you delay the presentation, the more you risk forgetting the implementation details and not being convincing enough that you understand the implementation.