Silesian University of Technology
Faculty of Automatic Control, Electronics and Computer Science

Computer Programming

Universal Electro Quiz Solver

author	Amadeusz Drożdż
instructor	mgr inż. Krzysztof Pasterak
major	Informatics
year	2018/2019
lab group	Tuesday, 12:15 – 13:45
date	July 2, 2019
github	**Project on github**

Table of contents

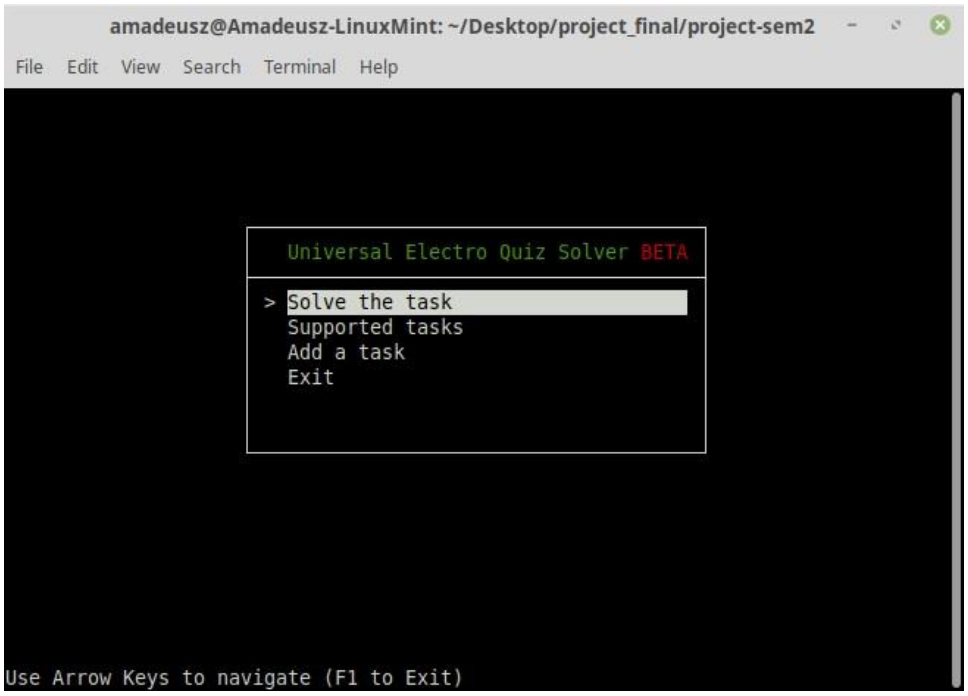
2 Project's topic

The goal of the project was to design a software that allows the user to quickly solve selected physics tasks using ready-made formulas previously added by someone else. Consequently, the user has the option to solve selected physics tasks and share his solutions with others. If the program supports the task requested by the user, it solves it using the correct formula and returns the result with appropriate unit.

3 Analysis and Development

I have uploaded a sample two minutes video to show clarify how the project works.

https://1drv.ms/v/s!AoXfPomWMn2tg9whSuJ67Ig4hPf8JA


amadeusz@Amadeusz-LinuxMint: ~/Desktop/project_final/project-sem2
File
Edit
View
Search
Terminal
Help

```
Universal Electro Quiz Solver BETA

> Solve the task
  Supported tasks
  Add a task
  Exit

Use Arrow Keys to navigate (F1 to Exit)
```

Figure 1: Main menu of the program

The full process is divided into three parts. First the user is adding a task to the database, then user solves one of the quizzes from “Fundamentals of Electric Circuits” course using the program and afterwards user checks all tasks added to database. On video all three options in the main menu are shown.

Main issues:

- Provide a method to simple and quick solving of tasks requested by the user
- Facilitating the adding process of tasks to the database, so that even not advanced users can perform it
- Fast and stable transfer and receive of the information with the server
- Reading user’s tasks (text analysis) with dynamic data
- Which libraries to use
- Which data structure to choose
- Error handling

The project is composed of many files and functions, description of all of them is included in Technical Documentation. Due to the complexity of the project, I had to use many external libraries. I used a GNOME GTK libraries hence the software is Linux platform only.

User interface:

- Ncurses (new curses)
- GTK 3+
- WebKitGTK

JS <> C callbacks: Apple JavaScriptCore

Regular Expression parser: Perl Compatible Regular Expressions (PCRE 2)

Mathematical expression parsing engine: tinyexpr

Https rest api requests: HISONA https_client

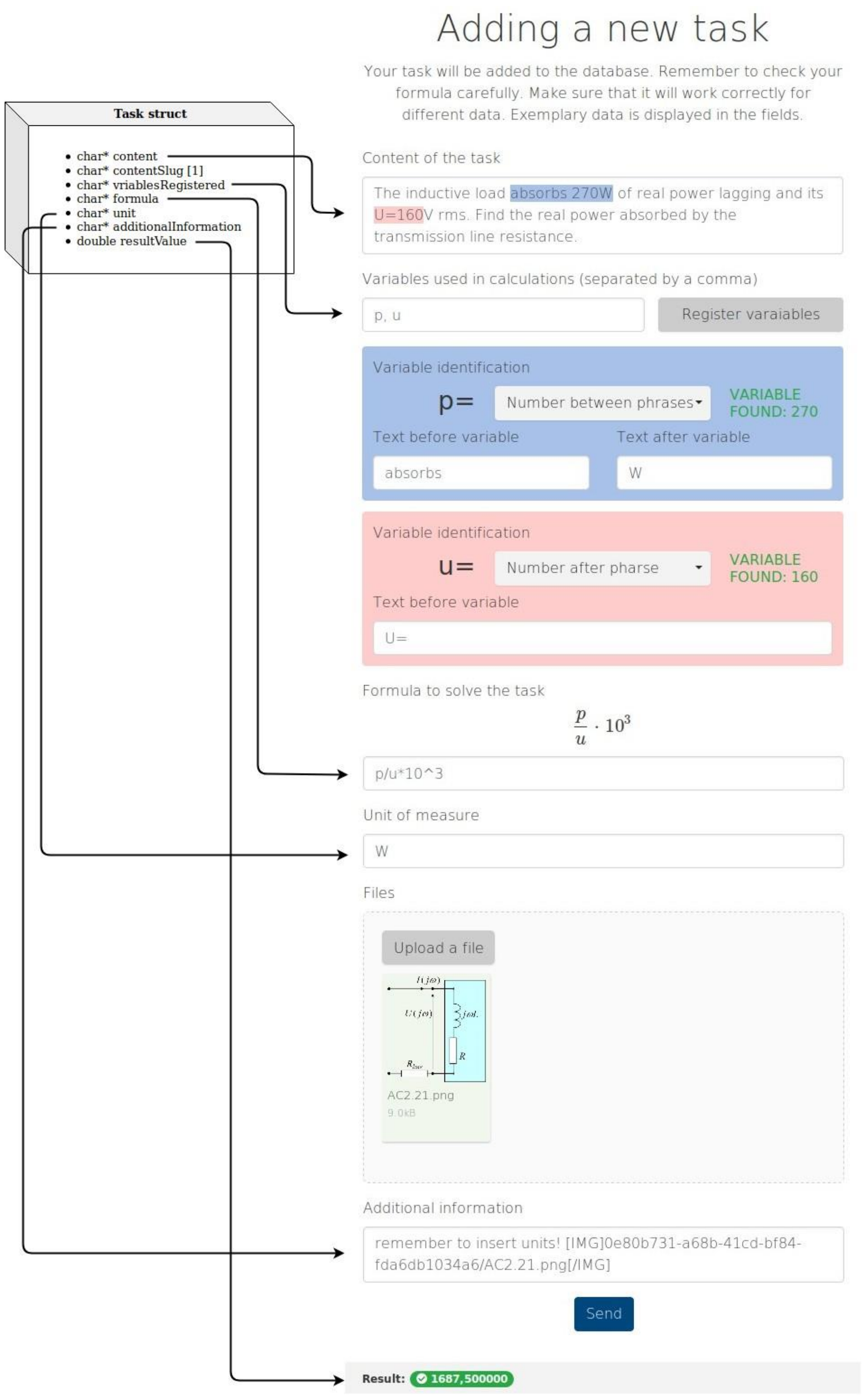
JSON parser: DaveGamble cJSON

Memory Allocation — general memory-handling: GNOME GLib (G Library)

Memory leaks detector: Memd

Because my main goal was the comfort of use (and I have needed advanced graphical user interface) I went with WebKitGTK browser engine, which needs to be implemented with GTK library. To create main menu which does not require that advanced gui I've used new curses library. So, when I decided to use website browser engine, I had to attain JS <> C commination, thus I chose JavaScriptCore with is compatible with WebKit framework. My program must also be able to extract data from the task content (text) and be capable of solving mathematical problems, so for that I used tinyexpr (Mathematical expression parsing engine) and PCRE 2 (Regular Expression parser). To obtain communication with the server and send data I've used: JSON parser to prepare the data to send and parse data received and https_client to create json REST request. For error handling I used GNOME GLib and memory leaks detector Memd.

3.1 Data structures



Adding a new task

Your task will be added to the database. Remember to check your formula carefully. Make sure that it will work correctly for different data. Exemplary data is displayed in the fields.

Content of the task

The inductive load absorbs 270W of real power lagging and its U=160V rms. Find the real power absorbed by the transmission line resistance.

Variables used in calculations (separated by a comma)

p, u

Register varaiables

Variable identification

p= Number between phrases

VARIABLE FOUND: 270

Text before variable: absorbs

Text after variable: W

Variable identification

u= Number after pharse

VARIABLE FOUND: 160

Text before variable: U=

Formula to solve the task

$$\frac{p}{u} \cdot 10^3$$

p/u*10^3

Unit of measure

W

Files

Upload a file

AC2.21.png
9.0kB

Additional information

remember to insert units! [IMG]0e80b731-a68b-41cd-bf84-fda6db1034a6/AC2.21.png[/IMG]

Send

Result: 1687,500000

Figure 2: "Task" structure representation

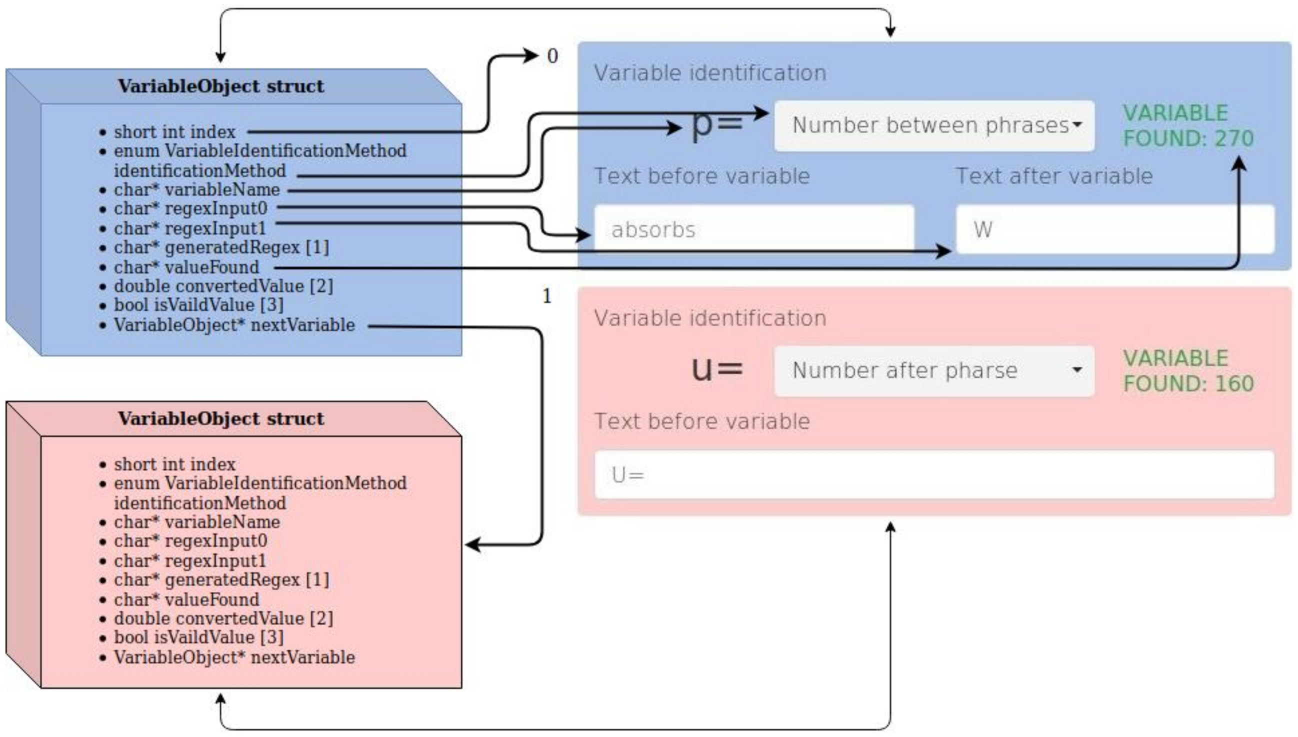
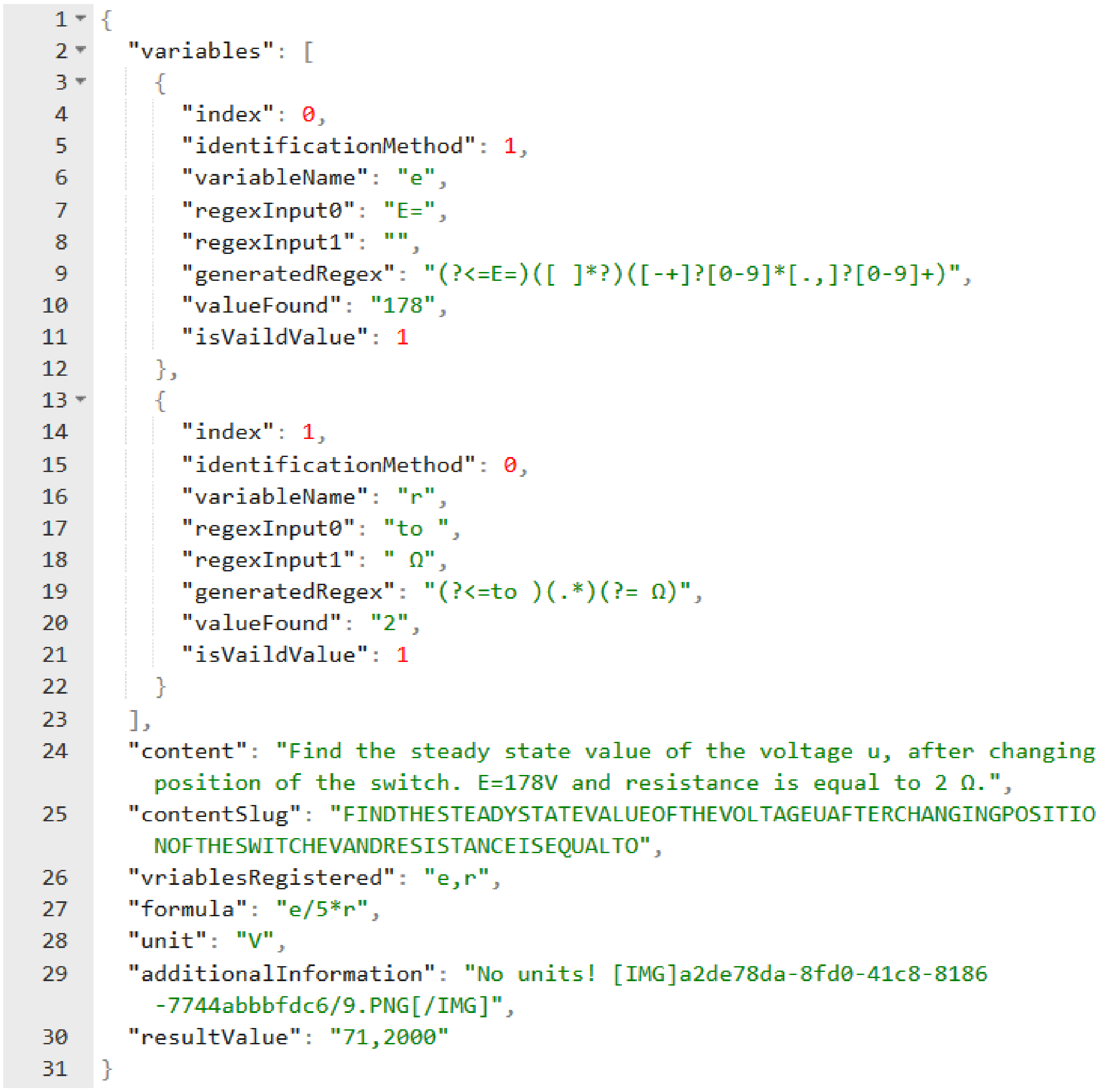


Figure 3: "VariableObject" singly linked list representation

When I created data structures for the program, I focused on the aspect that data structures would reflect as much as possible what the user sees on the screen. I needed to stone all user's input, for this reason I did not want to complicate the program unnecessarily. Thus, I have decided to use struct “Task” [Figure 2] to represent add data which are always single for a given task (add task request), such as task content or mathematical formula to get the task result. For the data like a variables used to solve the task which can be many I used singly linked list called “VariableObject” [Figure 3].

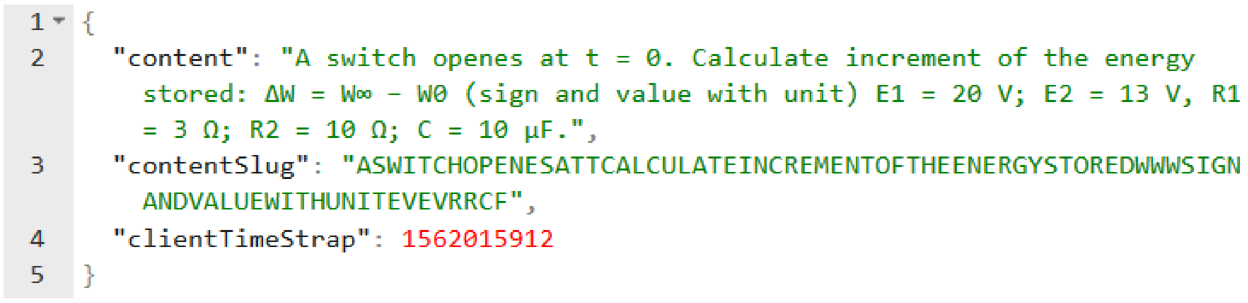
This is how exemplary task (rest api request) looks like when it's sent to the server (without headers eg. authorization):



```
{
  "variables": [
    {
      "index": 0,
      "identificationMethod": 1,
      "variableName": "e",
      "regexInput0": "E=",
      "regexInput1": "",
      "generatedRegex": "(?<=E=)([ ]*?)([-+]?[0-9]*[.,]?[0-9]+)",
      "valueFound": "178",
      "isVaildValue": 1
    },
    {
      "index": 1,
      "identificationMethod": 0,
      "variableName": "r",
      "regexInput0": "to ",
      "regexInput1": " Ω",
      "generatedRegex": "(?<=to )(.*)(?= Ω)",
      "valueFound": "2",
      "isVaildValue": 1
    }
  ],
  "content": "Find the steady state value of the voltage u, after changing position of the switch. E=178V and resistance is equal to 2 Ω.",
  "contentSlug": "FINDTHESTEADYSTATEVALUEOFTHEVOLTAGEUAFTERCHANGINGPOSITIONOFTHESWITCHEVANDRESISTANCEISEQUALTO",
  "vriablesRegistered": "e,r",
  "formula": "e/5*r",
  "unit": "V",
  "additionalInformation": "No units! [IMG]a2de78da-8fd0-41c8-8186-7744abbbfdc6/9.PNG[/IMG]",
  "resultValue": "71,2000"
}
```

Figure 4: Publish a new task - rest api request

This is how rest api request looks like when the user wants to solve the task (without headers):



```
{
  "content": "A switch openes at t = 0. Calculate increment of the energy
    stored: ΔW = W∞ − W0 (sign and value with unit) E1 = 20 V; E2 = 13 V, R1
    = 3 Ω; R2 = 10 Ω; C = 10 μF.",
  "contentSlug": "ASWITCHOPENESATTCALCULATEINCREMENTOFTHEENERGYSTOREDWWWSIGN
    ANDVALUEWITHUNITEVEVRRCF",
  "clientTimeStrap": 1562015912
}
```

Figure 5: Publish solve the task - rest api request

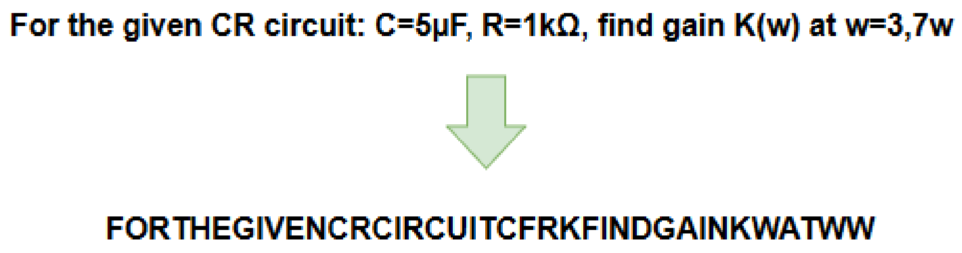
3.2 Algorithms and theory

When the user wants to add a new task, all form data is verified on the fly. Each text area changes causes calling JavaScript, which trough JavaScriptCore framework call C program functions, after performing the function in C language the value is returned into JavaScript method. In other words, by using JavaScriptCore I can expand JS language by my own C function. In the process of adding a new task there are three C functions triggered by the javascript:

- “validateTaskCb” - is used to save basic data about the task and put it into the “Task” struct, such as task content or formula to get task results.
- “findVariableCb” - is used to save information about one variable needed to calculate the task, this information is stored in “VaraibleObject” list.
- “submitTaskCb” - is called when user press the “Send” button, if the form is correctly completed the task will be send to server.

In turn, the task solving process requires only one function “solveTaskCb” which is called when user hits enter after giving the task content into the console. The “Supported tasks” part does not require sending any user data to the server; thus, it only loads website.

Then, once the user has already entered all the necessary data into the form. The data must be sent to the server, for that I used very popular JSON open-standard file format.



For the given CR circuit: C=5μF, R=1kΩ, find gain K(w) at w=3,7w

FORTHEGIVENCRCIRCUITCFRKFINDGAINKWATWW

Figure 6: the method of excluding dynamic variables

When you take a quiz form “Fundamentals of Electric Circuits” in every attempt you will receive another data in the tasks, so my program must recognize the given task regardless of the numbers that will change each time. That's why I added a “contentSlug” variable, it’s just a task content without any dynamic data; without any numbers and due to risk of incorrect text coding special characters are also removed. Removal of unwanted characters is done by regular expression search and replace with syntax “[^a-zA-Z]”.

To allow the program to extract data from the text I used regular expression which is a sequence of characters that define a search pattern. In program the user can define his own syntax or can use one of two pre-defined syntaxes. Available options:

- Number after phrase - (?<=)([ ]*?)([-+]?[0-9]*[.,]?[0-9]+) - number after given string
- Text between phrases - (?<=%s)(.*)(?=%s) - text limited by 2 given strings
- Custom regex expression - option for advanced users – custom regex expression in PCRE 2 standard.

The program is a typical example of client-server application. Application does not store any user data locally. All communication with the server is serviced using the HTTPS protocol

4 External specification

User’s manual.

4.1 Usage

The program requires the libpcre2 and webkitgtk-3.0 libraries to be installed on the user’s computer. In a Debian based Linux system this can be easily achieved by using package handling utility, running the two commands:

```
sudo apt-get install libpcre2-8-0

sudo apt-get install webkitgtk-3.0
```

Also, in the root directory of the program there is a script called “dependencies_installer.sh” the execution of which will install all required dependencies to run and successfully compile the program. The Makefile is attached thus, to compile the program user needs to just run “make” command.

To run the program, the user has to type the command below into their terminal:

```
./Universal_Electro_Quiz_Solver
```

The program should start, and user should be able to add new tasks and solve already added ones.

4.1.1 Keys

In main menu:

- F1 to exit
- Arrow Keys to navigate
- Enter to pick an option

In other options mouse as well as keyboard are supported.

4.2 Input data

Due to the specificity of the program, it is not required to give any data at the startup. The program consists of simple in use forms, which the user must complete to post data on the server.

4.3 Messages

The program prints helpful messages using web technology, informing
user about what might be wrong with the program and about possible errors in the form.

5 Internal specification

The program is implemented with structural paradigm.

5.1 Description of types and functions

Description of types and functions is moved to the appendix.

6 Running and testing

The program has been tested with various versions of Linux opening system, such as Debian, Mint and Ubuntu, after installing dependencies there was not any errors with running the program. The program uses leak detector called memd, the library did not find any memory leaks, which confirms me that there are no leaks. The program compiles without errors. While I was testing the software, I did not encounter any run time errors.

7 Conclusions

Writing this program was quite a challenge for me, but I'm very happy with the results, the program works exactly as I planned it. I encountered many difficulties while I was creating the software, the most severe for me was hardly available technical documentation in C language also managing the memory in the program needs a lot of care. I've learned a lot of new things while working on it like, for example: how to properly manage memory in the C applications, how to create applications of client-server type, how to create graphical user interfaces and many more.