8th International Scientific Conference
Technics and Informatics in Education
Faculty of Technical Sciences, Čačak, Serbia, 18-20th September 2020

Plenary Session

Keynote paper/Original research paper
UDC: 378:004.4

Program Translators Higher Education and Application of PP Simulator Educational Tool

Ljubica Kazi[1*], Dragica Radosav[1], Ivana Berković[1], Narendra Chotaliya[2], Madhusudan Bhatt[3]
[1] University of Novi Sad, Technical Faculty "Mihajlo Pupin", Zrenjanin, Serbia
[2] Saurashtra University, Rajkot, MP Shah Arts & Science College, Surendranagar, India
[3] University of Mumbai, K.C. College, India (retired)
[*]ljubica.kazi@gmail.com

Abstract: *Software engineering higher education usually includes courses related to program translators (compilers/interpreters), which cover topics: compiler construction, formal grammars, programming languages formal grammars and other formal representations (such as extended Backus-Naur Form), automata theory etc. Aim of this paper is to present overview of the current state in higher education of program translators and to describe the pragmatic approach that has been established at Technical Faculty "Mihajlo Pupin" Zrenjanin, Serbia with creating and using PP simulator educational tool. The developed tool helps students learn about the lexical, syntax and semantic aspect of programming code quality, which is to be determined by the compiler simulator "PP simulator". Teaching results from Technical Faculty "Mihajlo Pupin" Zrenjanin, Serbia at course Program Translators were also presented and discussed.*

Keywords: *program translators, higher education, educational tool, simulator, lexical analysis, syntax analysis, semantic analysis.*

1. INTRODUCTION

Following industrial trends, higher education needs to adapt in aim to enable students to have appropriate knowledge and skills needed for their professional engagements after graduation. During study time, students could also improve their knowledge/skills by being included in professional environments with internship and competitions organized by companies.

Modern software industry emphasizes agility of software development process and software product quality with having implemented constantly changing user requirements. These directions shift focus from using traditional programming languages to frameworks. Software frameworks and design patterns enable faster production and better quality of software products. Software frameworks are based on native (core) programming languages, but they have their specific grammars.

Regardless of native programming languages or frameworks usage, software quality particularly addresses detection and correction of possible code errors – lexical, syntax, semantic and run-time. In professional and educational programming environment, program translators (compilers, interpreters) are used for the purpose of translating programming code from higher programming language into machine code, while checking the code for errors.

This paper presents educational content, methods and results in teaching Program Translators (Serbian: name: "Programski prevodioci", abbreviated: PP) as a higher education course at Software Engineering bachelor studies at University of Novi Sad, Technical Faculty "Mihajlo Pupin" Zrenjanin, Serbia (abbreviated: @TFZR) in school year 2019/20.

The rest of this paper is organized as follows. Section two presents background and related work, section three describes higher education of program translators in Serbia, section four presents improvements of teaching Program Translators @TFZR with teaching content, methods, materials, assessment in school year 2019/20, section five describes PP simulator tool, section six presents teaching results including students experiments, while section seven presents conclusions.

2. THEORETICAL BACKGROUND AND RELATED WORK

2.1. Bloom's taxonomy

Benjamin Bloom (together with collaborators) published in year 1956 a framework for categorizing educational goals – Taxonomy of Educational Objectives, known as Bloom's taxonomy [1]. Initial Bloom's version explains main categories: knowledge, comprehension, application, analysis, synthesis, evaluation. Revised taxonomy has been performed by a group

of cognitive psychologists, curriculum theorists and instructional researchers, testing and assessment specialists in year 2001 [2] [3]. The authors of the revised taxonomy emphasize dynamism, by using verbs and gerunds to label categories and subcategories. This way, the attentions is drawn away from the "static" notion of educational objectives in Bloom's original title. "These action words describe the cognitive processes by which thinkers encounter and work with knowledge [3]: remember (recognizing, recalling); understand (interpreting, exemplifying, classifying, summarizing, inferring, comparing, explaining); apply (Executing, implementing); analyze (differentiating, organizing, attributing); evaluate (checking, critiquing); create (Generating, planning, producing). "In the revised taxonomy, knowledge is the basis of these six cognitive processes, but authors created a separate taxonomy of the types of knowledge used in cognition: factual knowledge, conceptual knowledge, procedural knowledge and meta-cognitive knowledge." [3]

2.2. Program translators

"On the very earliest computers, programs were written and entered in binary form. Some computers required the program to be entered one binary word at a time, using switches on the front panel of the computer. Because of that, the size and the complexity of the programs were severe limited, debugging was very difficult task and development of programs was very difficult and error prone. The idea was to use computer itself to ease the programmer's work, by translation from a more human-readable form of the program into executable binary code." [4]

There are several forms of program translation [4]: from assembly language code to binary coded instructions; from higher level programming languages to executable binary coded programs performed by compilers and interpreters; preprocessing with transformation from one higher level programming language to other lower level programming language, and then compilation is performed.

Programming languages grammars are presented usually with Extended Backus-Naur Form (EBNF) and syntax diagrams. ISO/IEC 14977 standard defines the elements and procedures of using EBNF [5]. Syntax diagrams [6] present graphical representation of programming language instructions syntax.

Programming languages include components [4]:

- Lexical – lexicon is a list of all legal words, together with information about the word (meaning, role);
- Syntax – define the form and structure of legal expressions of the language;
- Semantic – deals with the meaning of the expressions.

Formal grammars are used to present certain aspects of programming languages and translators, where Chomsky's linguistic-related research has created roots to this field [7].

2.3. Related work

Program translators' research has results closely tied with industrial advancements. In early years of higher-languages development, issues that researchers dealt with were related to special meta-languages for program translator construction [8], effectiveness [9], [10] and performances [11].

Recent program translators-related research and practical results deal with more advanced technologies, such as transformation from one higher language to another [12], multi-syntax programming languages [13] and embedded systems programming [14].

Research regarding programming languages education includes results in teaching formal languages [15] and comparative study of teaching two programming languages with the use of special tool for their automated translation [16].

3. HIGHER EDUCATION OF PROGRAM TRANSLATORS IN SERBIA

Higher education in Serbia in the domain of information technologies includes Program Translators at Bachelor and Master studies. In this section educational content will be presented from three Serbian Universities:

1. University of Nis, Faculty of Electronics – bachelor studies course [17].
2. University of Belgrade, Electro-technical Faculty – Bachelor studies course [18] and Master studies course [19].
3. University of Novi Sad, Faculty of Technical Sciences – bachelor studies course [20].

Educational contents at all faculties include: Formal languages, Automata theory, Lexical, Syntax an Semantic analysis of code, Symbol tables, Intermediate code generation and optimization, Code optimization, Program translators generators, Memory management, Virtual processors and machines.

The theoretical contents were illustrated at practical laboratory work by using simplified programming languages - MiniC [20] and MikroJava [18]. In aim to enhance their grammars, students use program translator generator tools, such as Flex [21], Bison [22], YACC [23] etc.

4. TEACHING AND ASSESSMENT IN COURSE PROGRAM TRANSLATORS @ TFZR in 2019/20

In aim to encourage students to be more active, gain more knowledge and have better results in assessment (comparing to previous years students' results), several aspects of education have been improved by the new subject professor Ljubica Kazi (assigned to theoretical and practical classes starting from school year 2019/20 @ TFZR):

- *Teaching content* has been put into a context of the modern industrial environment and other practical-oriented broader areas.
- *Teaching methods and knowledge/skills assessment* have been designed according to Bloom's taxonomy, particularly to be adaptable to students' abilities and preferences, as well as to improve interactivity with students during teaching and learning period.
- *Specific educational tool ("PP simulator")* has been developed as an educational tool - to enable students' experiments with compiler's work simulation.

4.1. Teaching content

In aim to enable students have better understanding of the abstract terms, defined as a core content of the course, the idea was to present the content within the broader modern industrial and practice-oriented context. This way, students could grasp the importance and usability of the presented theoretical concepts. Having this goal in mind, the focus on the most important topics of subject was not missed:

- Programming language grammar;
- Programming code errors;
- Compiler construction.

Background topics were presented at theoretical and practical lessons and linked with the core topic:

- Structured and object-oriented programming;
- Analysis and documenting of applicative software with UML;
- Software development with class libraries creation and linking;
- Programming integrated development environments;
- Abstract presentation of program specification (algorithms, flow diagrams, UML models).

Core theoretical topics included in teaching were organized as the sequence of logical flow, starting with topics previously familiar to students:

1. Types of programming languages, computer architecture, machine-dependent languages, definition of program translators;
2. Programming languages grammar – general linguistics and computer-based linguistics, formal languages and grammars, Chomsky's formal grammars categorization, forms of presenting programming languages grammars (EBNF, syntax diagrams);
3. Programming code errors – errors categorization (lexical, syntax, semantic, run-time);
4. Basics of functionality and construction of compilers – goals of compilers, compiler work phases, compiler architecture components, compiler working process variants, compiler components usability and functionality principles;
5. Automata theory – definition and categorization of automata, characteristics, using automata in language processing, Turing machine and Universal Turing machine.

Core practical topics in the laboratory work included:

1. Examples of compiler (C# desktop application with dynamic link libraries, i.e. class libraries) and interpreter work (PHP web application and XAMPP) in program errors detection, error types, messages, error handling, exceptions, validations of user inputs;
2. EBNF presentation of programming language grammar (C#, html, PHP)'
3. Compilation of structural and object-oriented code
4. Tools for program translator creation, creating, adjustments and using PP simulator as an educational tool.

Additional topics that were included in teaching content were selected in aim to put the core content into broader modern industrial and practice-based context:

- Code writing conventions and programming style,
- Test-based specification in agile software development,
- Domain knowledge and ontology languages, domain presentation, RDF,
- Software interoperability,
- Software frameworks and specific grammars, comparing native programming language and framework grammar,
- Cross-compilers,
- Linkers and module dependency,
- Software quality – standards, aspects (process, product, software in use), coding conventions and heuristics, programming style, code refactoring, software testing, agile test-based requirements specification, error processing in program code (Exceptions), principles of object-oriented program organization (SOLID), software performances, software metrics, structure aspect of software quality, code optimization.

4.2. Teaching methods and materials

Teaching period in school year 2019/20 for the course Program Translators (@TFZR) is, according to accreditation, planned for third year of study at Software Engineering Bachelor studies, for the "summer semester", starting in February and lasting to end of May. According to accreditation, students attend two teaching hours of theory and two teaching hours of practical laboratory work.

School year 2019/20 was specific due to appearance of Covid-19 (Corona virus) pandemic and the teaching period could be divided into 3 sub-periods:

1. *Pre-Corona virus lockdown period* – with regular classes, where students attended theory and practical work classes in classroom and computer lab. Theoretical classes were performed with oral and PowerPoint presentations (Figure 1), having theoretical and illustrative contents. In empirical part, practical work was performed as demonstrative analysis of software solutions, tools, grammars and related exercises that students were obliged to do with assistance of teaching staff. During this period, high level of interactivity has been performed at both theoretical and practical work and students' activities in discussions and solving problems were awarded with bonus points.

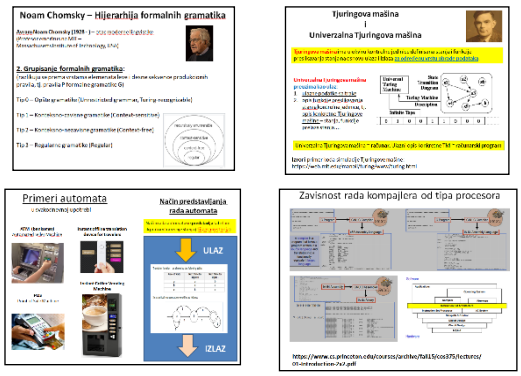

Figure 1. *Example of PowerPoint slides*

2. *Corona virus lockdown period* – online lessons, where electronic material (with theoretical content and practical content with illustrations of software solutions – source code of applicative software with class libraries and PP simulator, demonstrations with tutorials that explain the process and elements of solutions) was put on TFZR website at on-line teaching section within the course pages [24]. Students were offered to register at Facebook group PP@TFZR to receive frequent notifications about news and new materials that were submitted to on-line teaching section of Program Translators page at faculty website. Home works with practical orientation were included to replace the accreditation-defined "class attendance" points. To encourage students' interaction and activity, the aim was to have better learning outcomes by having students engaged during lockdown. There were two home work assignments:

   Homework 1 - input material was object-oriented applicative software (finalized version as continuing from regular practical classes) and the assignment was to create UML models to document the solution. Aim of this homework was to have students study the details of the applicative software, to understand and be able to make further analysis in next home work.

   Homework 2 - input was the same software solution as in home work 1, but with errors. The task was to start compiler and get report on errors, document lines of codes with errors, categorize errors, explain causes and correct them. The goal of second home work was to have students prepared for mid-term exam (with similar assignments).

3. *Post-Corona virus lockdown period* – elective classes, i.e. not mandatory classes. Students that wanted to attend, had to register for additional regular classes in classroom, with hygienic safety measures implemented. These classes were used for additional theoretical and practical explanations of on-line contents. These classes also had high level of interactivity with discussions, demonstrations of tools usage, students' questions and presentations of students' work.

4. *Students learning period* – work on their home works, projects and preparation for mid-term exam, as well as learning theoretical foundations. Practical learning work included empirical work with exercising in using, changing or creating program translators as tools, experiments with PP simulator work and other tools.

As a summary, during whole semester, *teaching methods* were selected in aim to increase students' attention and activity, so they included: presentations, illustrations, demonstrations, empirical work (experimental, practical work).

Special emphasize was put on preparation of *teaching materials*, which included:

- Power point presentations (examples presented at Figure 1);
- Theory text book;
- Handbook for practical work assistance;
- Simulation educational tool "PP Simulator" (Presented in section 5).

It is important to emphasize that theory text book was created to support only the core concepts of the course. Practical handbook included explanations, tutorial and examples for home

works, mid-term exam and project, as well as other topics included in core content. This way, students were given the essential source to prepare for all pre-exam elements and final theoretical exam. The books did not include additional content (background or additional industry/practical oriented content) in aim to avoid overload. The background and additional industry/practical oriented content were only presented during teaching time as illustrative for motivational and better understanding reasons. Knowledge in these fields was outside of the course boundaries, so the list of potential exam questions did not include these topics.

4.3. Knowledge/skills assessment methods

According to accreditation, course entitled "Program Translators" (PP@TFZR) includes three mandatory types of knowledge assessment:

- Mid-term exam (practical);
- Project (practical);
- Final exam (theoretical).

According to accreditation, in the grading points structure, the attendance at classes is also valued with certain points, but it does not encourage students to take active role in knowledge and skills development. Considering activity of students an important aspect of grading, additional bonus points were given to students that were collaborative in theory and practical work discussions or presented creativity and independence, preciseness and high level of details orientation during home works and regular class works.

Having enhanced Bloom's taxonomy as a starting point [1] [2], the assessment methods were designed at PP@TFZR to cover appropriate categories from the taxonomy (Table 1).

Table 1. *Categorization of knowledge/skills and appropriate assessment types at PP@TFZR*

Revised Bloom's taxonomy category	Type of knowledge/skills assessment at PP@TFZR
Create	Practical project
Evaluate	Mid-term practical exam
Analyze	Mid-term practical exam
Apply	Practical project
Understand	Mid-term practical exam Theoretical final exam
Remember	Theoretical final exam

Mid-term exam was organized to achieve pragmatic goal – to make students be able to detect program code errors, to classify them, detect causes and perform appropriate changes in aim to solve the problem.

Figure 2. presents an example of mid-term exam assignment, where students were given a software user interface, program code and compiler report about errors as input material.

The source code was the same one used for home works, but errors were made different. This way, students that regularly were active in home works could easily recognize code segments and benefit in faster tasks solving. Students task was, similar to second home work, to make classification of errors, explain them and provide solution with correctly written program code. Other type of assignment was to have correct program code and to make intentional errors of some type – lexical, syntax, semantic or run-time.

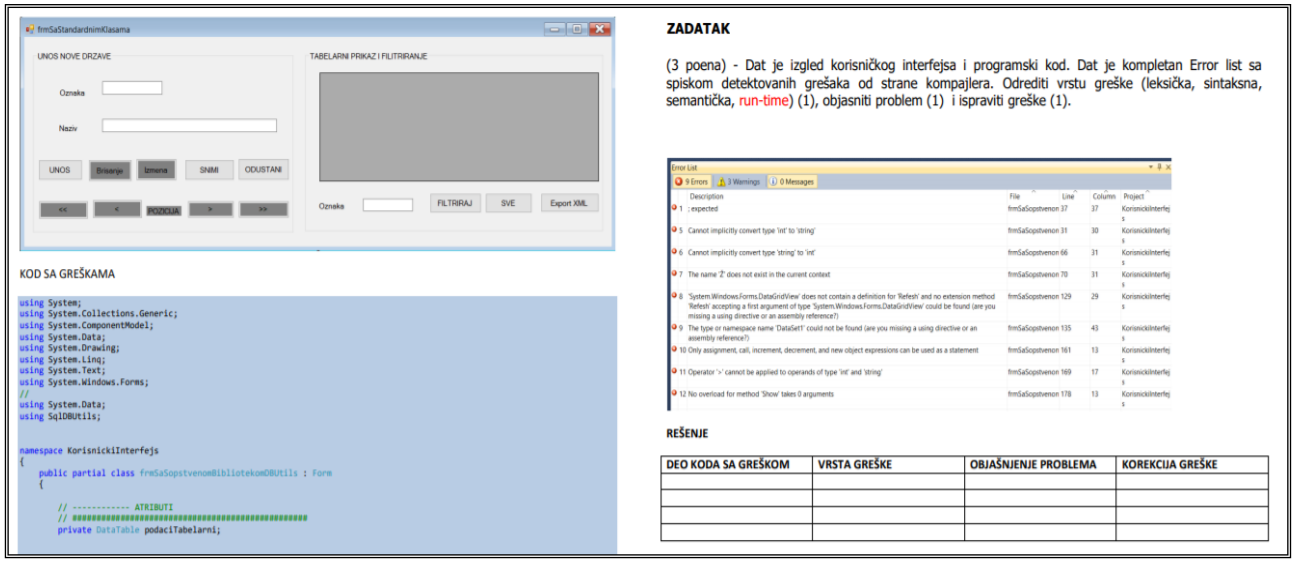

Figure 2. *Example of mid-term exam assignment at PP@TFZR in 2019/20 school year*

Project as a pre-exam practical work is designed to enable students to create, use or change a software solution in one of two mayor categories (elective):

1. Applicative software for certain problem domain
2. Compiler simulator that will have the functionality of analyzing lexical, syntax and semantic errors in program code segments or lines.

In aim to enable students to choose type of project (according to their self-estimation of knowledge, skills, available time, abilities, as well as their preferences/interests), there were 10 types of projects designed and offered to students to choose (with appropriate material and examples that are available for each type of project), as presented at Table 2.

Table 2. *Types of students' projects at PP@TFZR in 2019/20 school year*

TYPE	EXPLANATION OF PROJECT TYPE
1	Applicative desktop software, C#, using previously created Dynamic Link Library (DLL) for database connection and data operations, only changing an available example to different domain
2	Applicative desktop software, C#, using ready-made DLL for database connection, creating new DLL for data manipulation, changing an available example to different domain
3	Using compiler generator, for example GOLD [25], to analyze program code segment
4	Using compiler generator FLEX, BISON over the Mini C language, to expand the language grammar
5	Comparing grammars of two programming languages (e.g. C# and Java) with two applications over the same domain, with/without using database in the applicative software
6	Improving PP simulator to expand abilities to perform syntax and semantic analysis of program code – C# programming language
7	Improving PP simulator to expand abilities to perform syntax and semantic analysis of program code – other programming language (i.e. Java)
8	Creating program code analyzator for the programming language
9	Creating program code analyzator for the language used in programming (such as CSS, XML, JSON…)
10	Comparison of native programming language and framework with the example of applicative software

In any of these cases, software is created within certain development environment, which includes mandatory use of compilers - to detect errors and create EXE (executable file for desktop application) and DLL (Dynamic Link Library with classes) files.

As part of projects, students were assigned task to create errors intentionally, to have them categorized and corrected. If the PP simulator is chosen (project type 6 or 7), there were two types of errors to make intentionally:

1. Errors in program code line that represent an input to PP simulator (lexical, syntax, semantic);
2. Errors in PP simulator itself as an application (lexical, syntax, semantic and run-time).

5. PP SIMULATOR TOOL

During school year 2019/20 a tool for analysis of program line or segment has been developed by Ljubica Kazi at PP@TFZR. It was named "PP simulator".

PP simulator is able to analyze lexical, syntax and semantic aspect of quality of a program code line (Figure 3) and program code segment (Figure 4).

Work of PP simulator is based on:

- Predefined table of characters, that could be recognized as valid and categorized.
- Predefined table of words, i.e. character sequences that could be recognized and replaced with tokens.
- Predefined table of semantic patterns, considered appropriate syntax and semantic form.

In aim to make "PP simulator" work for particular programming language, it is necessary to have these tables filled with particular details related to the programming language grammar. This way, "PP simulator" is made ready to act upon the predefined grammar.

The process of "PP simulator" work and the principles of the tool function is described in sequence of automated actions the tool performs:

1. Recognition of characters and comparing with table of acceptable characters – lexical analysis.
2. Program line/segment reconstruction, eliminating blanks (space), line feed and carriage return symbols.
3. Recognition of words (lexeme) and comparing with table of acceptable words that could be replaced with tokens – lexical analysis (Figure 4). At the same time, recognized words are replaced with tokens and finally, the program code line is replaced with a tokenized sentence.
4. Comparing the tokenized line with syntax pattern and semantic pattern, determining if the tokenized equivalent of the program line/segment has been equal with any of the supported patterns. If the tokenized line matches with any of the patterns previously recorded, the line is considered correct. Otherwise, it is considered inappropriate for the previously defined grammar.

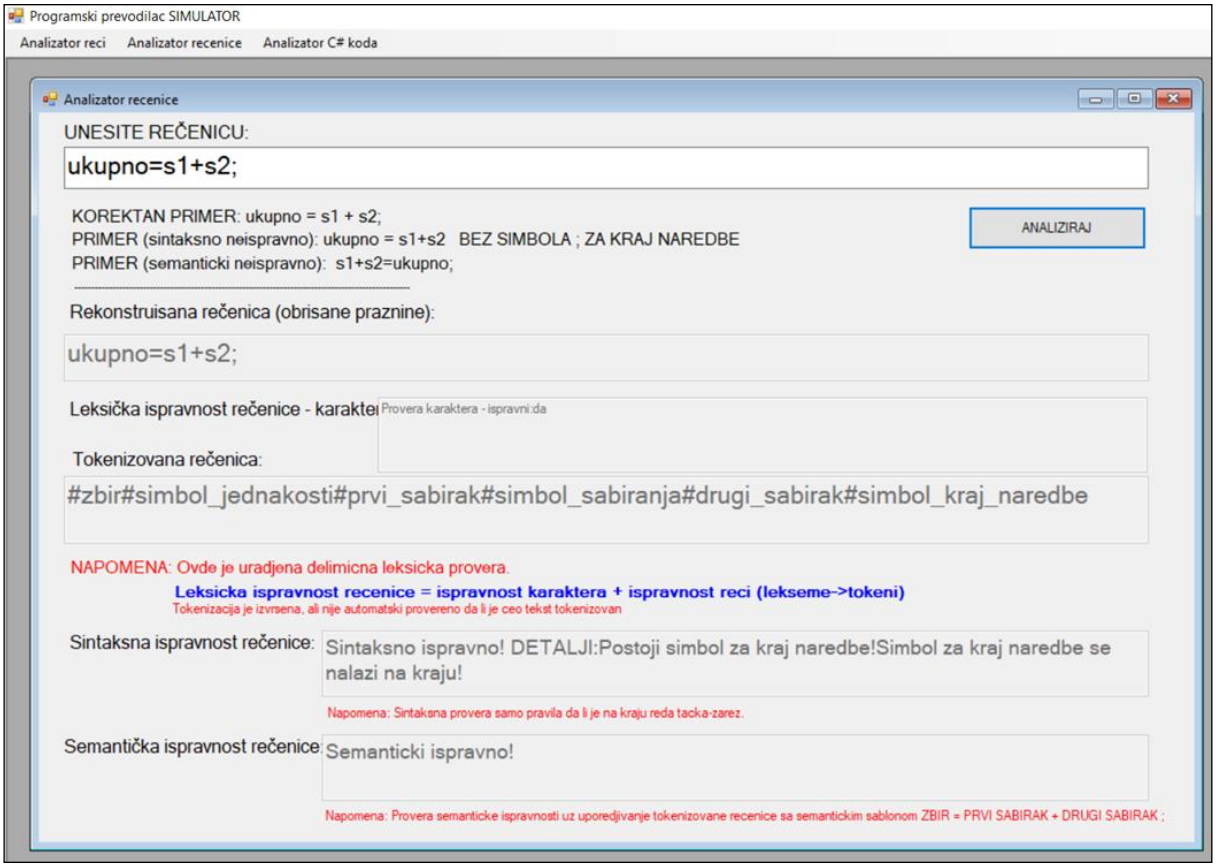

Figure 3. *Lexical, syntax and semantic verification of a program line in PP simulator*

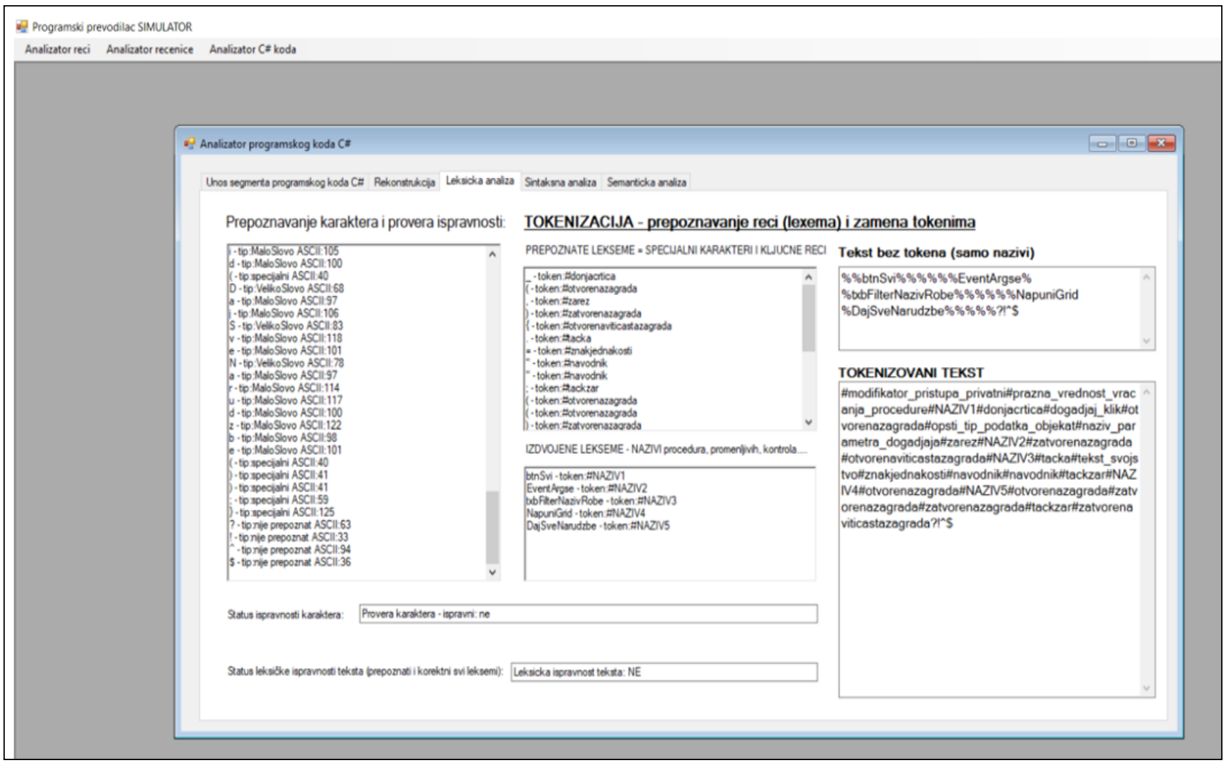

Figure 4. *Lexical analysis of incorrect code segment with recognition of character, words and tokenization in PP simulator*

6.TEACHING RESULTS

6.1. General teaching results

In this section the teaching results from the school year 2019/20 at PP@TFZR are presented. Research sample is based on 35 students results (complete number of 3rd year students in 2019/20 at the Program Translators course). The results are presented with the status on the end of semester (after the teaching session has ended) and after exams at June and July 2020.

Results are presented at Table 3 with class activity and number of students.

Table 3. *Teaching statistics at PP@TFZR in 2019/20*

Class activity	Number of students
Presence at regular theory classes	33
Actively work assignments at regular practical lab classes	27
Registered at Facebook group PP@TFZR (during lockdown)	24
Actively work on home works during pandemic lockdown	14
Registered and attending post-lockdown non-mandatory classes	9
Totally active at regular + lockdown + non-mandatory classes	34
Bonus for extra activity	24
Mid-term exam passed	24 (69%)
Project finalized	12
Passed whole exam	12 (34%)

In aim to demonstrate the effects of all efforts in improving teaching and learning environment for the course Program Translators @ TRZR in 2019/20, it would be beneficial to compare these results with teaching results from previous school years. It is important to mention that the course "Program Translators" started in school year 2017/18 with first generation of students.
In aim to have an approximately valid comparison, the teaching results will be presented for the same exam period, i.e. exams that were organized in June and July, immediately after the teaching semester for the course has been finished.

Data analysis is performed according to raw data available from the Program Translators pages @TFZR website [26].

Table 4. presents comparative data of general students' success for the exam terms June/July for three generation of students – school year 2017/18, 2018/19 with previous professor and, with new professor (having changed teaching goals, content, teaching methods, materials etc), generation 2019/20.

Table 4. *Comparative presentation of students' success for the PP@TFZR in three generation of students*

Year	Number of students in generation	Number of students that passed whole exam in June/July	% of students that successfully passed exam (June/July Exams)
2017/18	31	2	6
2018/19	36	2	6
2019/20	35	12	34

Figure 5. presents graphical representation of data provided in Table 4.

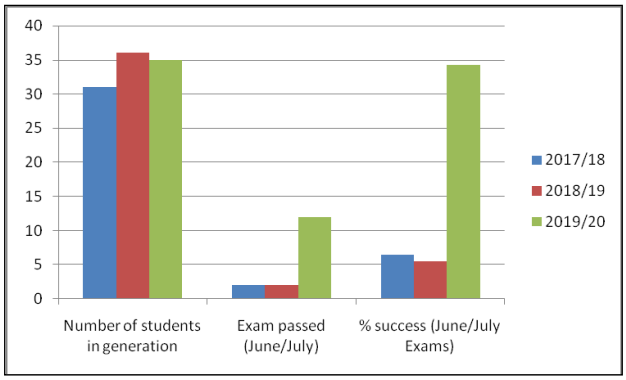

Figure 5. *Graphical presentation of comparative statistics of students success in three generations of PP@TFZR*

According to previously presented data, it is obvious that certain improvements have been made comparing to results from previous two school years period. Still, results for the overall students' success at first two exam terms (June, July) could not be considered satisfactory in 2019/20, since the whole exam passed only 34% of all students in generation.

6.2. Students' experiments in PP simulator-related projects

In projects in 2019/20 PP@TFZR, students used and changed PP simulator in aim to experiment (Figure 6, Figure 7, Figure 8, Listing 1 – authors are: student Bojan Babic with mentor Ljubica Kazi). Students used PP simulator for test code lines written in C# and Java programming languages.

In aim to make students aware of errors that compilers could detect (lexical, syntax, semantic) and run-time errors that could not be detected in compile-time, students were engaged, in their projects, to make intentional lexical, syntax,

semantic errors, as well as run-time errors. Example of making intentional run-time errors within the PP simulator tool is given at Figure 7.

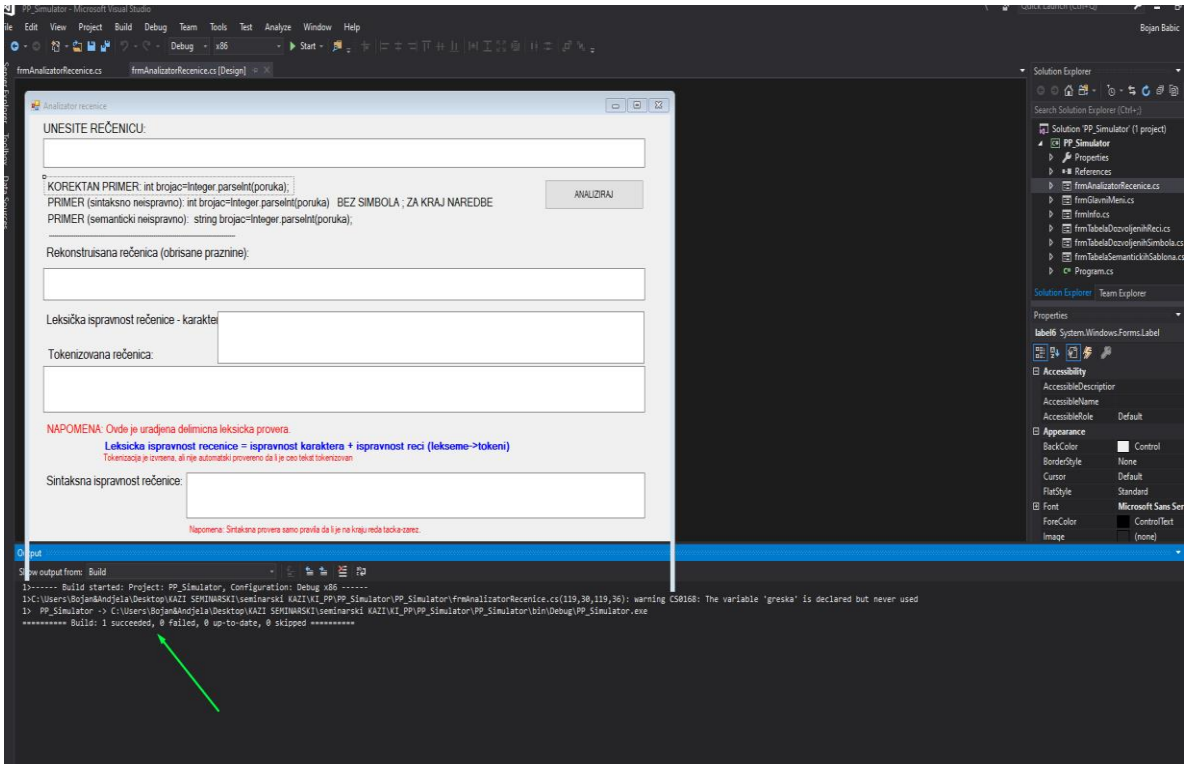

Figure 6. *Compilation result of the PP simulator as a software within Visual Studio NET*

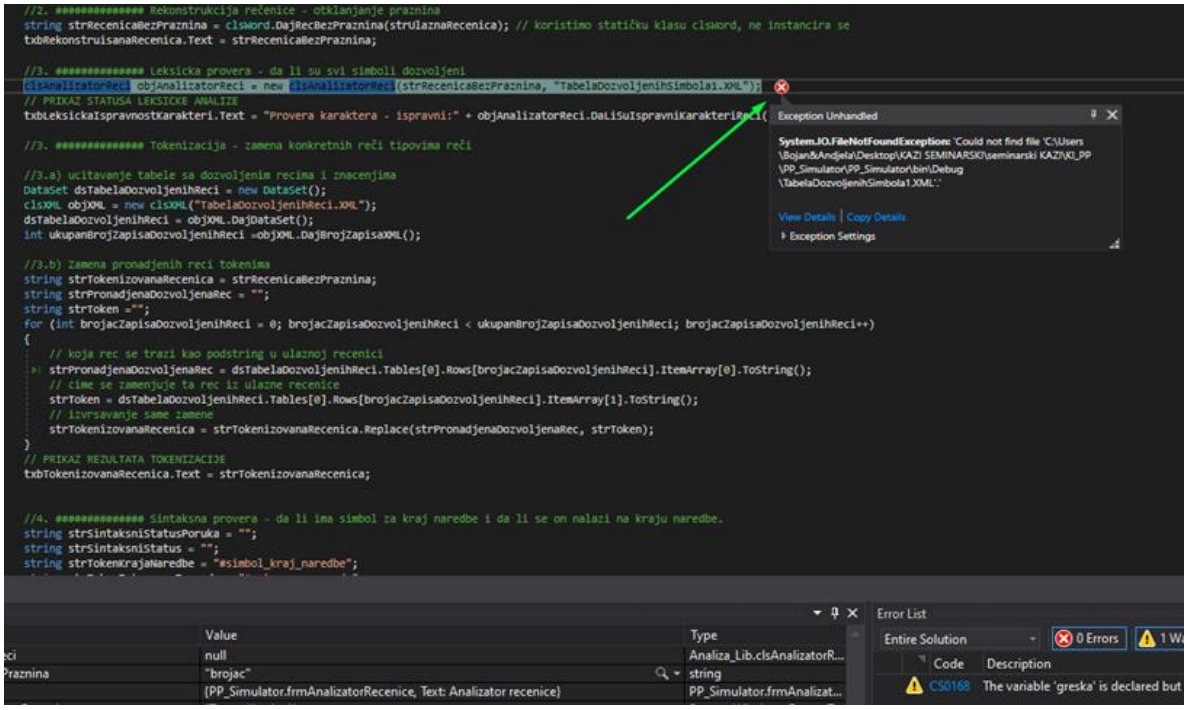

Figure 7. *Example of intentional run-time error in PP simulator*

Example of java program line to be tested in PP simulator with errors is given in Listing 1:

```
CORRECT
int brojac=Integer.parseInt(poruka);
LEXICAL ERROR
Int Br = Integer.ParseInt(poruka);
SYNTAX ERROR
int brojac=Integer.parseInt(poruka)
SEMANTIC ERROR
string brojac = Integer.parseInt(poruka);
```

Listing 1. *Example of experimental program line*

In aim to adjust PP simulator, code tables were updated to support lexicon and syntax/semantic patterns that are used for recognition and evaluation of the program line.

The error code lines and correct ones were put into the text box at the top and after starting the analysis, for the correct code line the result is given at Figure 8.

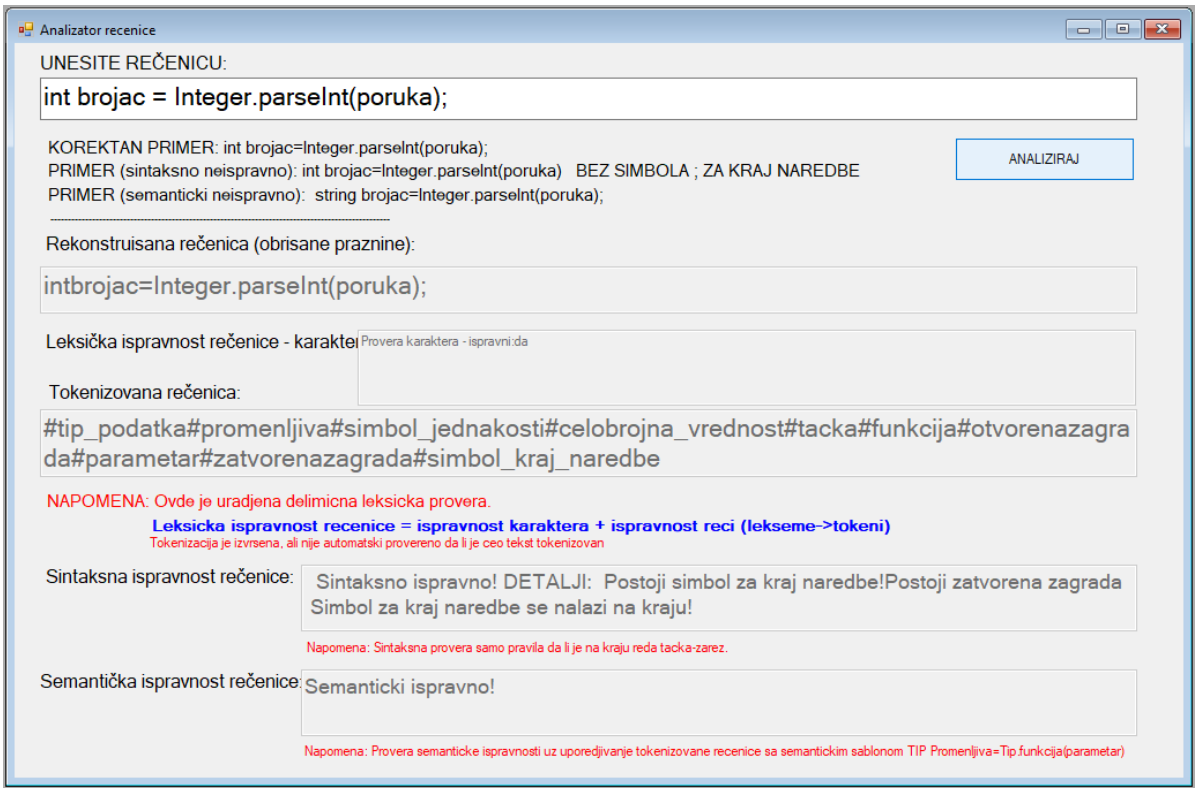

Figure 8. *Students project – results in experimenting with correct java program line in PP simulator*

7.CONCLUSION

Higher education constantly adapts to the needs of technology advancements, by including new teaching contents, but also new teaching materials, methods and tools. This paper presents improvements in teaching at course Program Translators at University of Novi Sad, Technical Faculty "Mihajlo Pupin" Zrenjanin, Serbia, made in school year 2019/20. Changes have been made in teaching content, teaching methodology and assessment, as well as in teaching material and educational tools ("PP simulator").

Aim of this paper was to present all the included changes, with particular emphasis on the developed tool "PP simulator". Finally, the outcome of all the efforts for the teaching improvements have been presented with teaching results statistics after first two exam terms (for the generation of students in school year 2019/20), compared to success percentage from previous two school years students.

General results of the presented teaching should be put in context of teaching process, materials, methods and tools, but also the specific situations with corona virus pandemic restrictions, which disabled students to attend regular classes in two months period starting from March 16 2020. It has been shown that students' interest to attend and actively participate in interactive regular classes has been much greater than doing home works during lockdown period. Finally, 69% passed mid-term exams, which shows that all improvement efforts made positive outcome.

Creating and documenting a software solution (with creating and handling errors with the use of compiler) is the essence of the students' project, which requires more time and effort. At first two exam terms 34% of all students have finished their projects and passed the whole exam. 25% of them chose to modify PP simulator.

Even the PP simulator has been designed as an universal tool for program code evaluation

(regardless programming language), it is expected in next years to be improved. Currently, it supports lexical, syntax and semantic evaluation of a code line and only lexical analysis and verification of a code segment. This version of PP simulator is based on syntax and semantic patterns which are compared with tokenized code lines in aim to determine their suitability. Of course, improvements should be made in this core principle of detecting the correct syntax and semantic forms of program code lines.

Having PP simulator closer to theoretical foundations of compiler constructions will enable students' better understanding of abstract concepts of formal grammars, automata theory and others. Having a better version of PP simulator will improve teaching environment in such way that it will encourage and direct students towards creating or modifying compiler simulators. This way, some of the project types will be excluded (such as 1 and 2), while those closer to compiler constructions will be emphasized.

Teaching content, methods, materials and tools are under constant improvements and adjustments to enable students have adequate knowledge and skills required in industry. In that context, it is very important to emphasize that, even new technologies and development environments encourage improvements in teaching process, the course core content should remain in focus, together with implementing academic principles of teaching and careful students' workload planning.

REFERENCES

[1] Bloom B, et al (1956): "*Taxonomy of Educational Objectives – The Classification of Educational Goals, Handbool 1 Cognitive Domain*", Longmans, Green and Co, LTD, London, David McKay Company

[2] Anderson L.W, et al (2001): "*A taxonomy for learning, teaching and assessing: a revision of Bloom's taxonomy of educational objectives*", Longman, New York

[3] Vanderbilt University Center for Teaching "*Bloom's taxonomy*", https://cft.vanderbilt.edu/guides-sub-pages/blooms-taxonomy/ [visited: 14th July 2020]

[4] Rautgerberg M: "*Lecture notes on Compilers"*, University of Technology, Eindhoven, https://rauterberg.employee.id.tue.nl/lecture notes/DA308/COMPILER.pdf / [visited: 14th July 2020]

[5] *ISO/IEC 14977 standard for Extended Backus-Naur Form*, https://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf / [visited: 14th July 2020]

[6] Gordon S: "*EBNF and Syntax Diagrams lecture notes*", College of Engineering, CSU Sacramento, *https://athena.ecs.csus.edu/~gordonvs/135/resources/05ebnfSyntaxDiagrams.pdf* [visited: 14th July 2020]

[7] Chomsky N (2006): "*Language and Mind*", Cambridge University Press.

[8] White J.R, Presser L (1973): "*A structured language for translator construction*", The Computer Journal

[9] Lucas H.C, Presser L (1972): "*A method of software evaluation: a case of programming language translators*", The Computer Journal

[10] Presser L, Benson J (1973): "*Evaluation of compiler diagnostics*", The Computer Journal, Vol 17, No 2

[11] Schneider V (1969): "A system for designing fast programming language translator", Spring joint computer conference AFIPS '69 proceedings, May 14-16, 1969, pp. 777–79

[12] McAtamney J (2010): "*C-to-Java programming language translator*", US Patent, https://patentimages.storage.googleapis.com/28/4a/52/d33e1b326637b5/US8533690.pdf *[visited: 14th July 2020]*

[13] Kuznetsov A.S. et al (2019): "*Enhanced pushdown automaton for recognizing multi-syntax programming languages*", Journal of Physics: Conference Series, ITBI 2019

[14] Ribic S (2006): "*Concept and implementation of the programming language and translator, for embedded systems, based on machine code decompilation and equivalence between source and executable code*", 13th IEEE Working Conference on Reverse Engineering

[15] Branstad D.K (1970): "*A computer-aided instructional system for teaching formal languages*", Iova State University, PhD thesis

[16] Li J et al (2017): "*Promotion of Educational Effectiveness by Translation-based Programming Language learning using Java and Swift*", Proceedings of the 50th Hawaii International Conference on System Sciences

[17] *Bachelor course of Program Translators at Elektronski fakultet Nis*, Serbia https://www.elfak.ni.ac.rs/downloads/akreditacija-2019/oas/rii/3OER7O03-programski-prevodioci.pdf *[visited: 14th July 2020]*

[18] *Bachelor course of Program Translators 1 at Elektrotehnicki fakultet Beograd*, https://www.etf.bg.ac.rs/sr-lat/fis/karton_predmeta/13S114PP1-2013#gsc.tab=0 *[visited: 14th July 2020]*

[19] *Master course of Program Translators 2 at at Elektrotehnicki fakultet Beograd*, https://www.etf.bg.ac.rs/sr-lat/fis/karton_predmeta/13M111PP2-2013#gsc.tab=0 *[visited: 14th July 2020]*

[20] *Bachelor course of Program Translators at Fakultet tehnickih nauka Novi Sad*, http://www.acs.uns.ac.rs/sr/pp *[visited: 14th July 2020]*

[21] Tool Flex (Princeton University), https://www.cs.princeton.edu/~appel/modern/c/software/flex/flex.html *[visited: 14th July 2020]*

[22] Tool Bison, https://www.gnu.org/software/bison/ *[visited: 14th July 2020]*

[23] Tool YACC, (Bell Labs and Stephen C. Johnson), http://dinosaur.compilertools.net/yacc/index.html *[visited: 14th July 2020]*

[24] On-line teaching page within Program Translators Course at University of Novi Sad, Technical Faculty "Mihajlo Pupin" Zrenjanin, Serbia, created by Ljubica Kazi in 2019/20, http://www.tfzr.rs/Predmet/programski-prevodioci/ucenje-na-daljinu-201920---elektronski-materijali-za-predavanja-i-vezbe *[visited: 14th July 2020]*

[25] Tool *"Gold"*, http://goldparser.org/getting-started/6-how-gold-works.htm [visited: 14th July 2020]

[26] Teaching results at Program Translators Course at University of Novi Sad, Technical Faculty "Mihajlo Pupin" Zrenjanin, Serbia, *http://www.tfzr.rs/Predmet/programski-prevodioci/ocene-predispitnih-obaveza-i-ispita* [visited: 14th July 2020]