Development of Web-based Visual Compiler for Computer Literacy

Hiroaki Hiranishi* Primož Podržaj Yoshiro Imai Tetsuo Hattori
Graduate School of Engineering, Kagawa University
Faculty of Mechanical Engineering in Ljubljana · University of Ljubljana
s17t260@stu.kagawa-u.ac.jp*,
primoz.podrzaj@fs.uni-lj.si
{imai.yoshiro, hattori.tetsuo}@kagawa-u.ac.jp

ABSTRACT

This paper proposes an educational tool, which can visualize the internal processing of the compiler. One of the purposes of this tool is to promote understanding of language processing and behavior of compiler from high-level programming language such as C into machine language (in our case, pseudo assembly language). This tool can run on major browsers of various operating systems because it is implemented in HTML5 and JavaScript. The internal structure is composed of the following four parts: input part for high-level programming language codes, lexical analyzing part, syntax analyzing part, and code generating part. With this visual compiler, we propose a model of computer literacy, which consists of 1) how to compile high-level language into related assembly code graphically, and 2) how to execute each assembly instruction by CPU simulator. We have demonstrated example lecture of computer literacy with our compiler and carried out questionnaire about lecture with compiler. And we report tentative results of questionnaire about our compiler.

KEYWORDS

Educational Visualization, Compiler, JavaScript, Questionnaire.

1 INTRODUCTION

We have to learn and teach programming in some computer languages from lower education to higher one. It is necessary for us to utilize suitable tools for effective learning of current computer programming. We have designed and implemented a Web application to learn and teach programming in C, which has been written in JavaScript.

C language is one of the most basic programming languages for Computer Literacy, which can provide code generation and cross compilation. This is meaning of compiling scheme, not of interpreting scheme. Simultaneously, however, we frequently face to the problem: Generated code can be really executed in CPU ? therefore, we have already developed CPU simulator as a(n) Web application as works for execution environment of generated code/instruction.

From our experience to utilize such CPU simulator, we need to use language processing facility of compiling service with visualization and Web-based scheme. The former of service is now really necessary for learners to understand how a compiler works graphically, while the latter of service is useful not only at practical classroom lecture but also at home and out of class.

This paper illustrates four major parts of our Visual Compiler, namely UI facility for source program direct description and selection of proposed examples, lexical analysis stage for extraction from statement of source program into lexical tokens, syntax analysis stage with creation of syntax tree from token table, and code generation stage which includes creation of symbol table and translation from statement of source program into pseudo assembly instruction. It describes demonstration of practical lecture for Computer Literacy through collaboration of Visual Compiler and CPU simulator. It also reports detail of questionnaire for trial evaluation of our compiler and its trial results from fresh members of company.

ISBN: 978-1-941968-62-8 ©2020 SDIWC

2 COMPUTER LITERACY MODEL

We classify model of computer literacy and propose a scenario of computer literacy.

2.1 Outline of Computer Literacy Model

Traditionally, outline of general computer literacy can be separated into a twofold aim: Utilizing computer and Understanding computer. The former deals with how to use major application software in order to apply PC with software into solution of practical problems. The latter includes learning computer organization, operating system and practical programming, especially in STEM education. Where STEM is frequently used as abbreviation of Science, Technology, Engineering and Mathematics. These are many famous textbooks about computer which are well-known and frequently used in the world: "Structured Computer Organization" by Andrew S. Tanenbaum[1] is one of the very famous textbooks including the assembly language level understanding in Chapter 7. That chapter explains importance of assembly language and assembly process practically, because assembly language has been considered to useful for learners to understand computer exactly. "Computer Systems – A Programmer's Perspective" written by Bryant and O'Hallaron[2] is also one of the famous and frequently used textbooks. This text provides general view of programming language and language processing which illustrates total scheme and organization of computer from high-level language (like C programming language) to executable object for computer.

Learning compiler is included in the subjects of above STEM education. Understanding compiler's internal processing is necessary not only to create a compiler but also to deepen comprehension of programming languages. For example, the 1st chapter of the above "Computer Systems – A Programmer's Perspective" introduces as follows: "The hello program begins life as a high-level C program because it can be read and understood by human beings in that form. In order to run hello.c on the system, the individual C statements must be translated by other programs into a sequence of low-level machine-language instructions. These instructions are then packaged in a form called an executable object program."

Aim of our project for computer literacy also can be separated into a twofold phase:

- How to compile the program
- How to execute the program

We will illustrate total view of our computer literacy and detailed role of compiler and its services for language processing.

2.2 Scenario of our Literacy

A scenario of lecture for our computer literacy is based on explanation of necessity of language processing and execution of its processed result by computer. Figure 1 shows image of our lecture of computer literacy, which brings up the problem how a computer recognize program, which can be understood

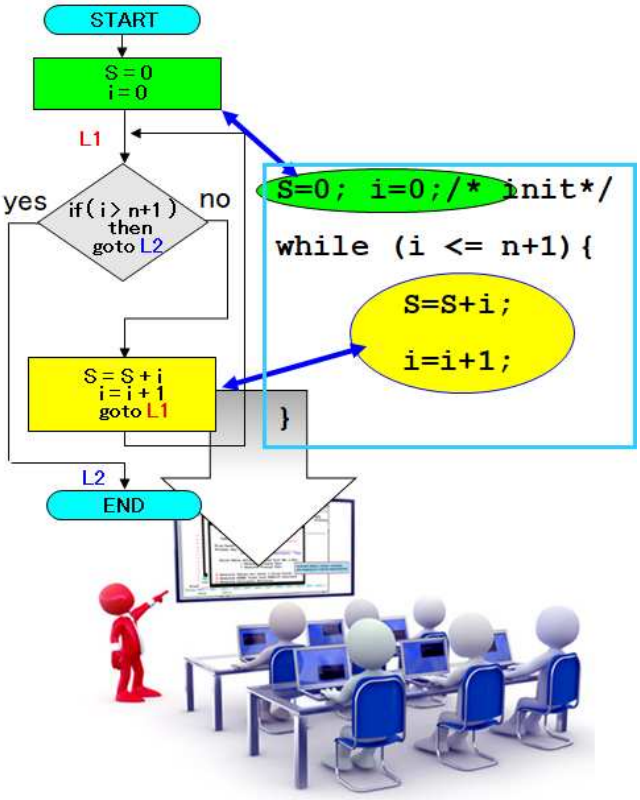

Figure 1. Lecture of Computer Literacy

by human beings, and how a computer executes/computes such program by itself. In order to realize such a lecture for Computer Literacy, we need to utilize a certain service of language processing which can handle program described in a high-level programming language like C, translation into executable object expressed in machine-specific instruction, and moreover to prepare the target computer which can execute generated assembly object directly for learners to understand behavior of compiler and computer.

One of our reviewers kindly pointed out that programming education can include several approaches we have to describe general explanation. "For students mostly trained in imperative programming (e.g., the C language) and object-oriented programming (e.g., the Java language), " some scenario of lecture for Programming educarion iused to be aimed at transitioning to the functional programming paradigm and /or to the logic paradigm[3]. And moreover, he kindly taught us to have to mention about programming environment on Web browser with 'WebAssembly'[4] because we had described our project about compiler on Web browser. WebAssembly can compile a variety of programming languages into WebAssembly executables, which can run on the major browsers through implementation of the corresponding virtual machines in Wasm. Wasm is a kind of standalone runtime environments built in major browsers.

We have developed Web-based educational tools for lecture of computer literacy, one of our tools is Web-based Visual Compiler and another is also Web-based Visual CPU Simulator. With these tools, we can provide several practical images and behaviors of language processing by compiler and illustration of execution of the processed results, namely executable objects, by CPU. Visual Compiler graphically explains how to translate from some program written in high-level programming language into a sequence of low-level instructions in assembly language in order for students to learn a role of compiler. Visual CPU Simulator shows how a computer performs such a sequence of low-level instructions also graphically.

With our compiler and simulator, we can propose a very specific scenario and graphical lecture of Computer Literacy not only for STEM education but also for introductory talk about computer science. The next section will focus on our Visual Compiler which treats with acquiring programs in C-like high-level programming language and generating executable object codes for CPU simulator.

3 WEB-BASED VISUAL COMPILER

This section describes design objective of our visual compiler, shows its user interface and illustrates multiple phases of its compiling behavior.

3.1 Design Objective

Generally speaking, in usual compiling processing, there are a sequence of sub-divided following parts of compilation, namely compiling behaviors. At the first phase of compilation, input program is to be parsed and processed at the lexical analyzing part of compiler. Generated lexical units is to be treated and converted into syntactical units at the syntax analyzing part of compiler. This is the secondary phase of compilation. At the final phase of compilation, code generation is to be performed together with syntactical units into target machine instruction.

Design objective to develop our Visual Compiler is aiming to provide clearly understandable roles for the above tree parts and illustration of three phases with their mutual correlation for the sake of visual compiler education. Visual Compiler has been designed and implemented in HTML5 and JavaScript in order to be easily utilized on the major browsers for user's convenience and operability. Users, namely learners of compiler, will be able to confirm how a compiler works graphically, at the lexical analyzing phase, syntax analyzing one and code generating one. For the sake of Learner's deeper understanding, the compiler must be designed and implemented in order to

ISBN: 978-1-941968-62-8 ©2020 SDIWC

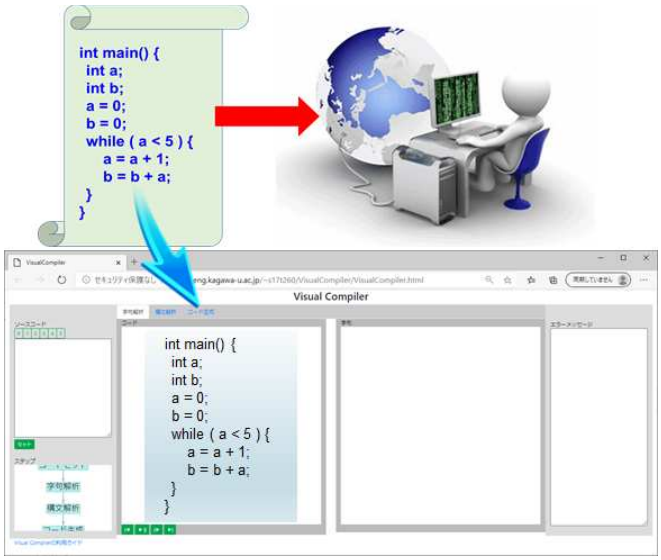

Figure 2. User Interface of our Visual Compiler

operate each phase of it individually while each phase has to cooperate with each other.

3.2 User Interface

It is important for learners to do compiling programs written in high-level programming language by themselves. In order to understand behavior and role of compiler, learners had better use compiler actually to apply a sample program in high-level language, to translate it into executable object, and to verify whether such object actually executable or not. User interface has been designed carefully for learners to recognize detail behavior of compiler, to manipulate three phases directly and to obtain intermediate and final results of compiling process. Figure2 shows user interface (namely UI) of our Visual Compiler and example of source program for compilation.

Learners prepare their source programs which can be executed by computer. They lay them on the input code area of UI of compiler directly for compilation. Of course they can select some example source codes on the leftmost column of UI in Fig2. They are prepared by compiler for learners to easily start to understand how a compiler works graphically. Learner's source codes sometimes include wrong expression so that compiler generates report of error message on the rightmost column of UI.

3.3 Four Phases of Compiling

Source program is to be parsed and processed at the lexical analyzing part of compiler. This part generates lexical units, called 'token', and the relevant parse tree for the next phase. The syntax analyzing part can provide special visualizing services how syntax analysis can be performed with lexical units and parse tree, because it does specify and highlight the close-up leaf of parse tree corresponding to the selected lexical unit in the manner of mouse-clicking by users. Code generation part provides a sequence of low-level machine-language instructions translated from the individual C statements.

Our Visual Compiler translates source program written in high-level language into low-level assembly code through lexical phase, syntax phase and code generation phase. They are summarized as follows;

1. Lexical Analysis:"generation of lexical token table"
   Figure3 shows a view of lexical analysis of Visual Compiler. With Visual Compiler, learners specify the lexical analysis phase to scan each statement of source program, extract lexical units called 'token' from statement and generate token table for the next phase. They can choose menu of lexical analysis to generate token table in the manner of step-by-step mode based on mouse clicking or in the manner of automatic style. The former will be useful to understand closely detail of lexical analysis, while the latter is suitable for learners to obtain the results of lexical analysis immediately.

2. Syntax Analysis: "generation of syntax tree"

   Figure4 shows a view of syntax analysis of Visual Compiler. Learners specify the syntax analysis phase to create the corresponding syntax tree from statement of source program and the token table generated at the lexical analysis Such syntax

ISBN: 978-1-941968-62-8 ©2020 SDIWC

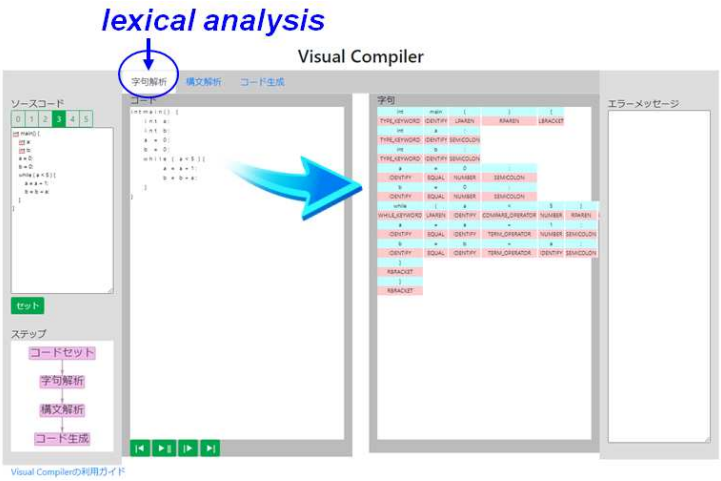

Figure 3. Lexical Analysis of Visual Compiler

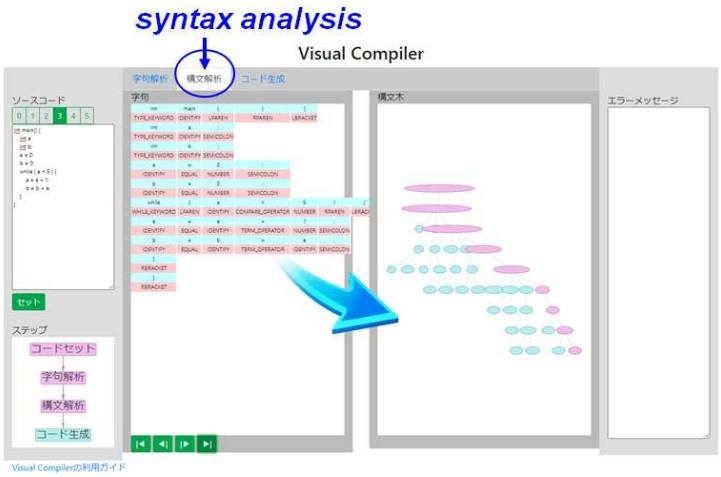

Figure 4. Syntax Analysis of Visual Compiler

tree is growing up from the top (root of the tree) to the bottom (leaves of tree) as referring the source and the token table. This tree can be smoothly enlarged easily to look at the detail of the tree. For example, zooming up and zooming down of syntax tree facilitates useful ways to check the leaves of the branch that correspond to each statement and to look at the entire tree at a glance, respectively. Learners can also choose menu of syntax analysis to create the syntax tree in the manner of step-by-step and/or automatic modes for their suitable understanding.

3. Code Generation(1):“generation of symbol table”
   Figure5 shows a view of code generation of Visual Compiler. The first half of this phase is dedicated to generation of symbol table and the second half is for code generation. The symbol table has been generated through scanning of every leave of syntax tree and located below the syntax tree shown in Fig.5.

4. Code Generation(2):“generation of assembly code”
   Code generation has been performed based on syntax tree and symbol table. Generated code has been located at the right side of syntax tree and symbol table shown in Fig.5. Learners can choose menu of code generation in the manner of step-by-step and/or automatic modes for the sake of learner’s convenience. Such code of executable object is expressed in a specific form of assembly instructions that can allow learners to recognize how such code instruct computer more easily than pure binary code.

3.4 Verification of Compiled Object

After compilation, learners need to confirm whether generated code can be executable correctly or not. Therefore our scenario of lecture for computer literacy has provided another facility for the sake of verification of compiled object. Figure 6 illustrates how to verify practical evidence of executability of compiler-generated code.

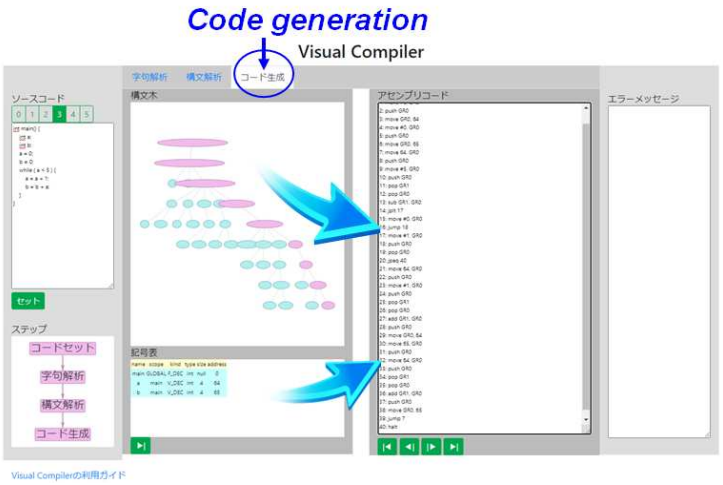

Figure 5. Code Generation of Visual Compiler

ISBN: 978-1-941968-62-8 ©2020 SDIWC

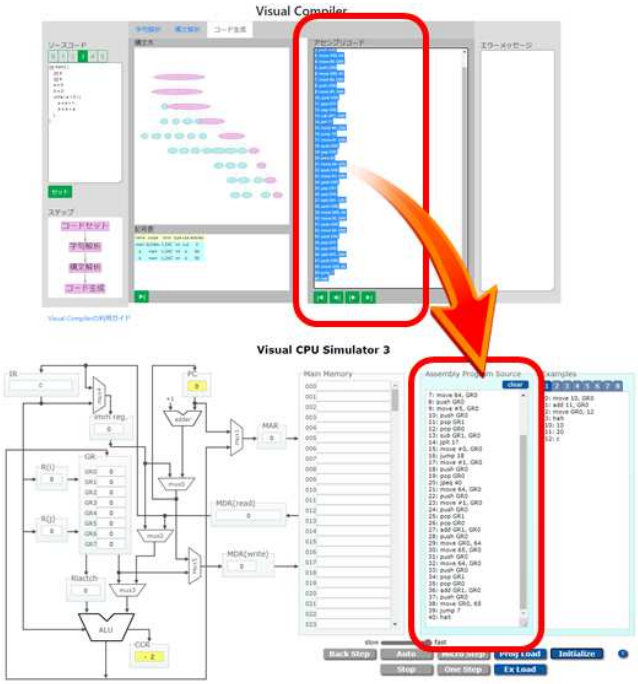

Figure 6. From Compiler to CPU Simulator

Learners can specify the generated code, move such code between Visual Compiler and Visual CPU Simulator by means of manner of "copy and paste" style as shown in Fig.6. In other words, generated code can be reproduced from the upper compiler of Fig.6 to the lower simulator of Fig.6 in the simple method. That means loading executable object generated by compiler into the memory of computer, namely the specific fram corresponding to the memory

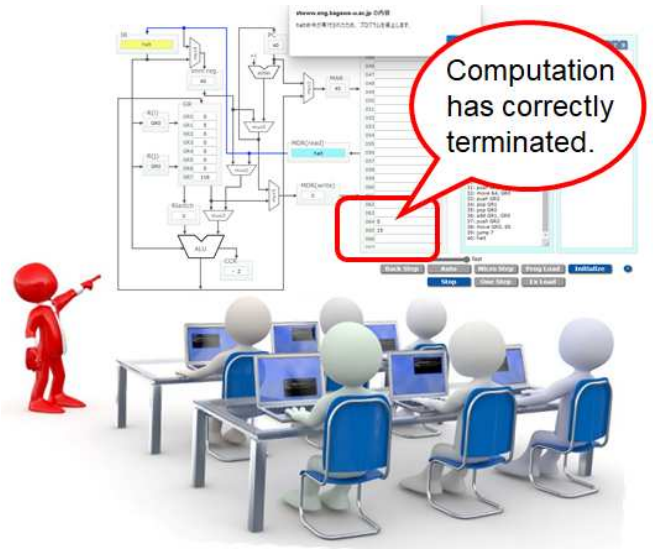

Figure 7. Demonstration of Program Execution

of Visual CPU Simulator.
Our Visual CPU Simulator[5] can execute assembly program in the following manner. Our Simulator has been written in HTML5 and JavaScript also in order to be easily utilized on the major browsers. Figure 7 shows how a computer works correctly with Visual CPU Simulator and verify how to execute given assembly program by our simulator.

- step-by-step execution of given assembly source:
  Visual CPU Simulator provides step-by-step execution of machine-cycle level, namely executing assembly program in the stepwise manner of each assembly instruction using button for "one step execution". This execution style is suitable for learners to check and confirm how the program works.

- automatic execution of given assembly source:
  The simulator also provides continuous execution in the repetition mode of machine-cycle level, namely automatically executing assembly program until the end of program using button for 'automatic execution". This execution style is suitable for learners to obtain the result of program immediately.

4 EVALUATION OF COMPILER

This section explains the contents of questionnaire and its results as evaluation of our compiler based on user feedback in the classroom lectures.

4.1 Contents of Questionnaire

After usage of our simulator in the practical classroom lecture, we had carried out the following seven numbers of questionnaire for our learners who were the first year students of our departments. Learners are asked to choose five options from Grade5(Excellent) to Grade1(Poor) for each question.

(1) Is the User Interface(UI) suitable for you to utilize our Visual Compiler (our system) ?

ISBN: 978-1-941968-62-8 ©2020 SDIWC

(2) Is it effective for you to understand how to extract token through lexical analysis with our system ?

(3) Is it effective for you to understand how to create syntax tree through syntax analysis with our system ?

(4) Is it effective for you to understand how to generate object code with our system ?

(5) Is it useful for you to utilize our system in order to understand how to translate from source program into executable codes ?

(6) Is it effective for you to understand relation between translation by compiler and execution by CPU together with Visual Compiler and CPU Simulator, respectively ?

(7) Please describe some possible merit about our system, if you can find out.

(8) Please point out some items to be improved about our system.

(9) Any comments and suggestions will be always welcomed.

We have carried out questionnaire about our Visual Compiler for about 10 members of relatively small company which has provide image processing Camera and system, after explaining and demonstrating our Visual Compiler in approximately 20 minutes. And we have received 7 feedback from listeners of explanation and demonstration of compiler and simulator. The aim of the above questions are to confirm whether characteristics and merits of our Visual Compiler can provide user's benefits and contribution to learner's understanding.

4.2 Results of Questionnaire

Results of the above questionnaire is summarized in bar-chart graph of the Figure 8. Evaluation of our Visual Compiler based on the above results for questionnaire has been summarized as follows;

- Because of the results of questions (2) and (6), namely approximately more than 60% of answers are agreeable for the relevant questions, it may be confirmed that our Visual Compiler can achieve understanding major stages of lexical analysis, syntax analysis and code generation of compiler.
- However, because of the results of questions (1), UI of our Visual Compiler has been considered to not so good. Therefore, we have to modify of UI of compiler and improve its performance based on the below actual comments and suggestions in order for learners to manipulate our compiler efficiently and effectively.

Typical comments from Questionnaire is summarized as follows:

- I think it's very nice that it has a function that points out code errors. The person who writes the code will notice his mistake, and the person who starts learning the code will notice that "If you write this, you will get an error". When it comes to complicated circuits, the more basic mistakes are, the harder it is to notice. As I mentioned at the meeting, the layout of each area seemed to have room for improvement. In the lexical analysis area, there was a place where the width was cut off, so I wonder if it's okay to widen the side and display the code a little narrower. The lexical analysis part was visually easy to grasp and understand, but I could not grasp the syntax tree and the process of

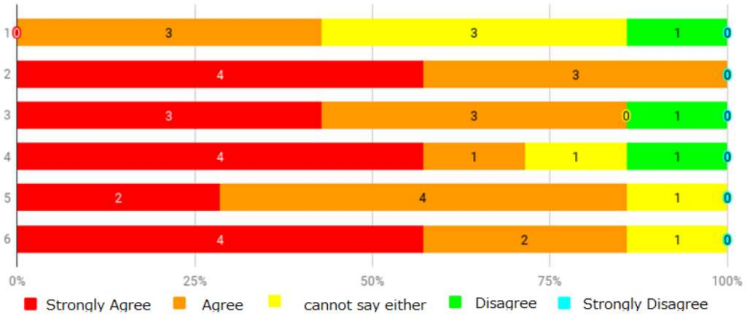

Figure 8. BarChart graph for the result of questionnaire: Red for Grade5 · · · Lightblue for Grade1

ISBN: 978-1-941968-62-8 ©2020 SDIWC

generating the objective code. For those who don't know at all, a syntax tree was created at the push of a button.

- I felt that it would be very helpful for understanding.

- I like the fact that I can understand the basic concepts of the compiler. I thought it would be better to refine the UI (user interface) a little more.

- I felt that it was easier to imagine because I could perform step execution. It seems that the size of the GUI can be improved a little more. I think it is effective to use both of Visual Compiler and Visual CPU.

- I've ever seen assembly code, but I've never seen the syntax tree, so it was very interesting for me. I think the letters were small. I thought it would be nice to know at a glance the correspondence between source code and assembly code.

- By using Visual Compiler and Visual CPU Simulator in combination, I thought it was good to be able to visually teach program beginners the essential processing hidden by high-level languages. I thought there was room for improvement in the UI.

- When training beginners in programming using this system, if I am an instructor, I think it would be easier to explain if I show the lexical / syntax tree (preferably the assembler) together. I felt that this system was very useful as a teaching material for beginners to understand the essence of how the program works.

5 CONCLUSION

We have developed a Web-based educational tool to learn language processing practically, which can provide the detail of compilation including the lexical analysis, the syntax analysis and the code generation. This tool called Visual Compiler has been develped to provide useful educational material for Computer Literacy and to be applied into demonstration of compiler and actual practice in the exercise. Our Visual Compiler can run on the major browsers due to writing it in JavaScript entirely. Therefore, learners can utilize it very easily and understand how a compiler works graphically.
We have proposed a scenario of lecture for Computer Literacy with Visual Compiler, which can explain how to compile program and how to execute program simultaneously. Such a lecture has been already performed for the beginners, namely fresh persons of some company, and feedback from learners have been also obtained through lecture. Questionnaire has been carried out after finishing our lecture so we have approximately good feedback from the learners but we have to improve User Interface of our compiler due to evaluation result from the learners.

REFERENCES

[1] Andrew S. Tanenbaum. Structured Computer Organization. Pearson, Prentice-Hall, 1990

[2] Randal E. Bryant and David R. O'Hallaron. Computer Systems. Pearson Education, 2015.
https://dadongwang.files.wordpress.com/2011/03/computer-system-a-programmers-perspective.pdf

[3] A. Bossard and K. Kaneko. A new methodology for a functional and logic programming course, Proceedings of the 20th ACM Annual Conference on Information Technology Education (SIGITE), pp. 63-68, Tacoma, WA, USA, October 2019.

[4] Haas, Andreas; Rossberg, Andreas; Schuff, Derek L.; Titzer, Ben L.; Gohman, Dan; Wagner, Luke; Zakai, Alon; Bastien, JF; Holman, Michael (June 2017). "Bringing the web up to speed with WebAssembly". Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation. Association for Computing Machinery: 185–200. doi:10.1145/3062341.3062363

[5] Shinya Hara and Yoshiro Imai. Register-Transfer-level CPU Simulator for Computer Architecture Education and Its Quantitative Evaluation, IEEJ Transaction on Electronics, Information and Systems, 138(9), 1123-1130, 2018. https://doi.org/10.1541/ieejeiss.138.1123