A Proposal of Code Writing Problem for C Programming Learning Assistant System

Htoo Htoo Sandi Kyaw[1,*], Chai Xu Min[1], Keiichi Kaneko[1], Soe Thandar Aung[2], Nobuo Funabiki[2], and Annisa Anggun Puspitasari[3]

[1] Department of Computer and Information Science, Tokyo University of Agriculture and Technology, Tokyo, Japan
[2] Department of Information and Communication Systems, Okayama University, Okayama, Japan
[3] Department of Intelligent Mechatronics Engineering, Sejong University, Seoul, South Korea
Email: htoohtoosk@go.tuat.ac.jp (H.H.S.K.); s216754t@st.go.tuat.ac.jp (C.X.M.); k1kaneko@cc.tuat.ac.jp (K.K.); soethandar@s.okayama-u.ac.jp (S.T.A.); funabiki@okayama-u.ac.jp (N.F.); annisanggun@gmail.com (A.A.P.)
*Corresponding author

***Abstract*—C programming has been a fundamental subject for a lot of university students studying programming languages, algorithms, and computer architecture. To enhance C programming education in schools, we have developed the C Programming Learning Assistant System (CPLAS) which provides a variety of programming assignments to cover different learning stages. The programming assignments offered by CPLAS allow the students to practice writing partial source code to learn grammar, code reading, and code debugging skills. However, the current assignments offered by CPLAS do not cover students to practice writing the whole C source code from scratch. Therefore, in this paper, we propose the Code Writing Problem (CWP) for students to write the C source code from scratch. In a CWP instance, a problem statement, an input list, and an expected output are given to the students. The students need to write the C source code from scratch by referring to the given information. The answer codes are marked through 1) compiling test, 2) execution test, and 3) output test. To avoid cheating by students, we use the random input generation method for each data type. To evaluate the proposal, we generated and assigned 11 CWP instances to the students. From their solution results, 10 out of 11 CWP instances achieved over 78% correct answer rate, and the feedback from the students confirmed the validity of the proposal.**

***Keywords*—C programming learning assistant system, C Programming Learning Assistant System (CPLAS), code writing problem, CWP instance**

I. Introduction

C programming maintains its pivotal status as a foundational programming language. Across global academic institutions, numerous universities incorporate C programming, along with its object-oriented extension C++, as the inaugural computer programming language. This practice extends beyond IT departments and extends to diverse fields such as mechanical engineering and electrical engineering.

Furthermore, within IT departments, the study of C programming frequently coincides with computer architecture courses. This dual approach stems from the necessity to comprehend memory and register access by programs, ensuring efficient programming within the context of the specific computer architecture. Consequently, despite its longevity since its inception, C retains its position as the second most prominent programming language due to its enduring relevance and widespread applicability [1].

Therefore, to assist C programming education for students, C Programming Learning Assistant System (CPLAS) has been developed as a self-study tool for students. CPLAS provides different programming assignments to cover different levels of students. Presently, CPLAS provides simple exercise problems, such as Grammar-Concept Understanding Problem (GUP) [2], Value Trace Problem (VTP) [3], Element Fill-in-Blank Problem (EFP) [4], Code Completion Problem (CCP) [5] and Phrase Fill-in-blank Problem (PFP) [6], listed here from easiest to hardest, respectively.

In CPLAS, firstly, a GUP instance consists of a source code and a set of questions describing the definitions of important elements in the source code. By solving GUP instances, students are expected to master the basic grammar concepts of C programs. Secondly, a VTP instance asks students to trace the value of a variable to make them master code tracing. Thirdly, an EFP instance consists of a source code with blank elements where the students are asked to fill in the correct answer for each blank aiming to improve their code reading skills. Fourthly, a CCP instance asks for the completion of the given source code by filling in the missing elements. By solving CCP instances, students can master the code debugging ability. Finally, a PFP instance asks students to fill in the phrase in the source code. This PFP instance allows the students to practice writing the partial source code.

The assignment problems mentioned above are mainly focused on grammar, code reading, and code debugging study. To master the C programming language, students need to write the source code from scratch and all the current programming assignments offered by CPLAS do not cover acquiring the code writing skills. The

Manuscript received June 20, 2023; revised August 10, 2023; accepted October 30, 2023; published April 11, 2024.

programming assignment which allows the students to practice writing the source code from scratch after studying the grammar, code reading, and code debugging is needed to be included in our CPLAS.

Due to this motivation, for the students to study writing the whole source code from scratch, we propose the Code Writing Problem (CWP) as a new type of programming assignment in CPLAS. A CWP instance consists of a problem statement, an input list, and an expected output. A student needs to refer to the given information in a CWP instance and write the whole source code from scratch.

To mark the students' answers, we use the code validation function [7] to the students' answer source codes. The code validation function validates the answer source code through 1) compiling test, 2) execution test, and 3) output test. The first two steps are done by the gcc compiler for C programs. For the output test, the output from the students' source code and the output from the correct answer source code are checked using the Levenshtein distance. To avoid cheating by students, we randomly generate the input data and are given to the students' source code and correct answer source code for the output test.

For the evaluation, we generated 11 CWP instances and asked 9 students from the Tokyo University of Agriculture and Technology and 1 student from Niigata University to solve. The answer results and feedback from the students confirmed that CWP is valid as a new programming assignment in CPLAS to study writing C source codes from scratch.

The rest of this paper is organized as follows: Section II proposed the code writing problem in CPLAS. Section III evaluates the proposed code writing problem. Finally, Section IV concludes this paper with future work.

II. Proposal of Code Writing Problem

In this section, we present the proposal of the code writing problem.

A. *Overview of CWP*

In a CWP instance, a problem statement, an input list, and an expected output are given to the student. A student writes a C program source code from scratch by referring to the given problem statement and the expected output. The code validation function checks the student answer code through 1) compiling test, 2) execution test, and 3) output test. Compiling test and execution test are done by gcc compiler while the output test is done by matching the output of the student answer with the output of the correct answer using Lavenshtein distance. Here, in a CWP instance, the problem statement is fixed however the input list and the expected output are randomly changed every time a student submits the answer. This is to prevent cheating from the student by simply just copying the expected output in their answer code.

B. *CWP Instance Generation Procedure*

The following four steps show the generation steps to generate a CWP instance:

(1) Select a source code from a website or a textbook and register it as a model answer source code.
(2) Create the problem statement and register it.
(3) Specify the input file condition to randomly generate the input list.
(4) Apply the code validation function to generate the input list and expected output.

C. *Implementation of CWP on the CPLAS Platform*

In this section, we will present the implementation of CWP on the CPLAS platform using Node.js [8], Express.js [9], and EJS.

1) Software architecture

In this particular implementation, the server platform operates on Ubuntu 22.04 LTS, running on Windows Subsystem for Linux (WSL) on the Windows 11 operating system. The web application server is implemented using Node.js, in conjunction with the Express.js framework. EJS serves as the designated template engine for this system. Instead of employing a dedicated database, all data management operations are carried out using a conventional file system. The GNU Compiler Collection (GCC) is utilized as the C language compiler in this setup. Editing of the source codes is performed using the Visual Studio Code (VS Code) integrated development environment. Within this architectural design, the model (M) component relies on Python to handle all subprocesses related to the compiler, while the view (V) component incorporates EJS, CSS, and JavaScript. Finally, the controller (C) component leverages JavaScript, specifically Node.js and Express, to fulfill its role.

2) Sever-Side implementation

Node.js exhibits a sophisticated and intricate architecture, rendering it challenging to maintain. Consequently, in this particular implementation, *Express* is adopted in conjunction with Node.js. Express.js adheres to the MVC (Model-View-Controller) structure, which serves as a programming design pattern. Utilizing Express requires the installation of both Node.js and the accompanying node package manager (npm) through the designated installer for each respective operating system. npm assumes a pivotal role in facilitating Node.js application development by granting access to a comprehensive assortment of reusable JavaScript libraries. These libraries are vital for diverse stages of application development, encompassing development, testing, and production. Furthermore, npm enables the execution of tests and the utilization of development tools during the software development process.

The application environment utilizes npm to import additional dependencies required for its operation, including frameworks and template engines. In this application, the dependencies that are being used include bootstrap, codemirror, date-utils, debug, ejs, express, http-errors, and so on.

Upon successful installation of Express.js, the system generates several directories, namely bin, nodemodules, public, routes, and views. Subsequent sections will provide comprehensive elucidation on the functionality and significance of each directory in facilitating the optimal operation of the applications.

Within the Node/Express.js framework, individual web applications are created and executed on dedicated web servers. Express.js offers mechanisms for explicitly defining the appropriate function to be invoked for a specific HTTP [10] verb (e.g., GET, POST, SET) and URL pattern, commonly referred to as a "route". Additionally, Express.js facilitates the specification of the designated template "view" engine, along with the precise location of template files and the specific template employed to generate a response for rendering purposes.

3) Connection between server and browser

The interaction between the server and the browser occurs through the following sequence of procedures when a client accesses the server's designated URL (Uniform Resource Locator) with the specified port, such as "localhost:2000", from the browser. This process is illustrated in Fig. 1.

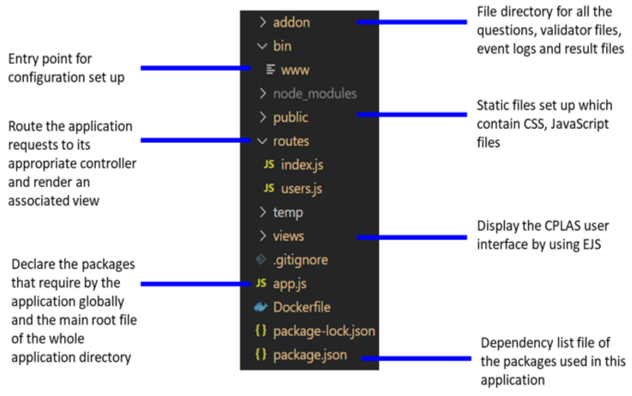

Entry point for configuration set up
Route the application requests to its appropriate controller and render an associated view
Declare the packages that require by the application globally and the main root file of the whole application directory
addon
bin
www
node_modules
public
routes
index.js
users.js
temp
views
.gitignore
app.js
Dockerfile
package-lock.json
package.json
File directory for all the questions, validator files, event logs and result files
Static files set up which contain CSS, JavaScript files
Display the CPLAS user interface by using EJS
Dependency list file of the packages used in this application

Fig. 1. CPLAS application directory structure using Express.js.

- package.json: Initially, the web application references the package.json file. This file serves as a comprehensive record of dependencies associated with a specific JavaScript "package". It includes essential details such as the package's name, version, description, initial file for execution, production, and development dependencies, as well as the compatible version of Node.js. The package.json file encapsulates all the necessary information required by npm to retrieve and execute the application. Within the file, the "start" key, and its corresponding value, "node/bin/www", are located in the script tag. This configuration signifies that the Node.js project invokes the "www" file located in the bin folder, which effectively collects the data indispensable for the functioning of Express.js within the application.
- www: The "www" file serves as the entry point for the application and enables the configuration of various setup parameters. In the current application, three distinct scripts are configured within this file: app, debug, and http.

4) Adopted languages for MVC model for CWP

The MVC model serves as the adopted software architecture for the Code Writing Problem (CWP) within the CPLAS platform. This model adheres to the standard architecture employed in web application systems. Fig. 2 presents an overview of the software architecture designed for addressing the Code Writing Problem. In this specific configuration, Python is employed within the model component, whereas EJS (Embedded JavaScript), CSS (Cascading Style Sheets), and JavaScript are utilized for the views component. JavaScript alone fulfills the role of the controller component in this MVC-based software architecture.

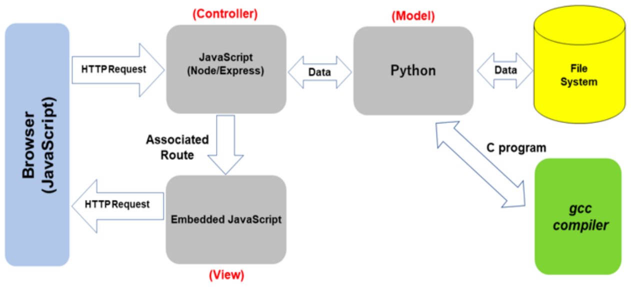
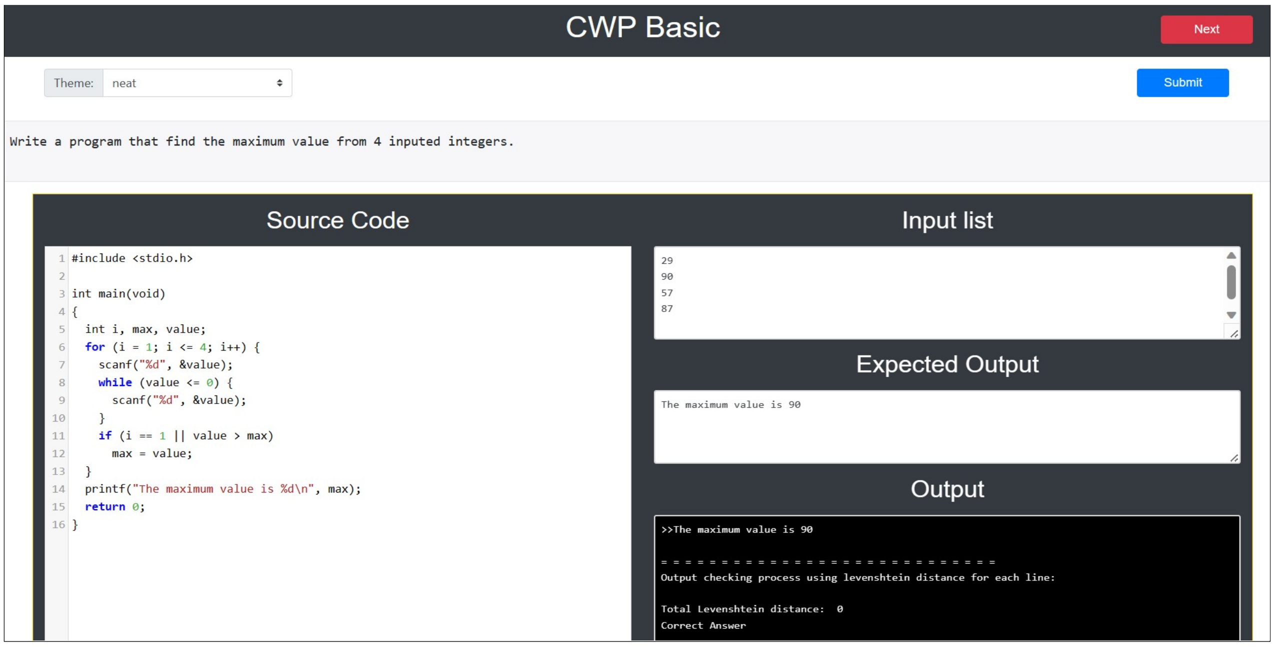


Fig. 2. MVC model in CPLAS using Node.js.

- Model: The model component in the CPLAS is responsible for accessing and processing essential data files stored in a predefined file system known as "addon". These files consist of student assignment answer code files and validator files. The marking function within CPLAS varies based on the problem type. Specifically, for the code writing problem, Python is utilized to implement the marking function.

  During the marking process, the function generates randomized input and subsequently produces the expected output by executing the model answer code. If the student's answer code is successfully compiled, the function proceeds to compare its output with the expected answer output, employing the calculation of the Levenshtein distance. The marking results, including the compile log, are then recorded in the file system and made accessible to the student through the browser interface.

  To ensure data persistence, the student's answers, and the corresponding marking results are temporarily saved in the browser's local storage using a unique key. Subsequently, students have the option to store this data from the local storage into a text file within the output folder, located in the "addon" directory of the application.
- View: The view component in the CPLAS assumes the responsibility of implementing the user interface. This is accomplished through the utilization of the EJS template, enabling the dynamic rendering of content within the browser. To enhance visual aesthetics, the SkyBlue CSS framework is employed for styling purposes.



Write a program that find the maximum value from 4 inputed integers.

```c
#include <stdio.h>

int main(void)
{
  int i, max, value;
  for (i = 1; i <= 4; i++) {
    scanf("%d", &value);
    while (value <= 0) {
      scanf("%d", &value);
    }
    if (i == 1 || value > max)
      max = value;
  }
  printf("The maximum value is %d\n", max);
  return 0;
}
```

```
29
90
57
87
```

```
The maximum value is 90
```

```
>>The maximum value is 90

= = = = = = = = = = = = = = = = = = = = = = = = = = =
Output checking process using levenshtein distance for each line:

Total Levenshtein distance:  0
Correct Answer
```

Fig. 3. Problem answer interface for code writing problem.

EJS is configured to construct HTML code along with JavaScript code snippets, which are dynamically passed to the application's backend. Specifically, for the code writing problem, the CodeMirror CSS is utilized to facilitate syntax highlighting, gutters, and certain keyboard shortcuts. All EJS files pertaining to the application are located within the designated "views" directory.

The application initiates the rendering process by accessing the index file, which serves as the home page of the application. This file incorporates essential components such as the title, menu, and main body, each tailored to the specific problem type. Fixed aspects of the interface are generated using EJS and CSS, while the variable sections are generated through JavaScript functions stored in the public directory.

To ensure modularity and simplicity within the code architecture, the main body of the interface is dynamically exchanged with the relevant EJS file based on the client's requested route. This approach reduces code redundancy and streamlines the overall code structure.

- Controller: The controller component within CPLAS is implemented using JavaScript. Upon receiving a request, the application proceeds to determine the appropriate action based on the URL pattern and the associated information conveyed through POST or GET data. This may involve accessing or modifying data stored within the file system, as well as performing any necessary tasks to fulfill the request. Express.js facilitates the seamless transfer of data through the designated routes. During data transfer, the route is assigned a relevant name, which aids in organizing the flow of information.

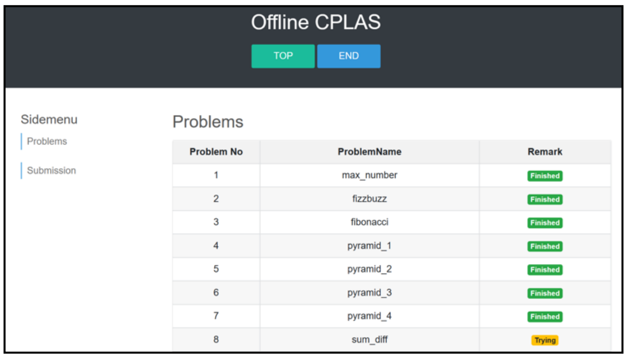
  Subsequently, the application generates a response to be sent back to the web browser. This often entails the dynamic creation of an EJS page, which serves as the view to be rendered within the browser. The generated page is tailored to the assigned name and incorporates the retrieved data by inserting it into the appropriate placeholders within the EJS template.

  Node.js and Express.js collectively possess the capability to effectively handle websites with dynamic data. Furthermore, the hierarchical structure and grouping concepts inherent in Node.js and Express.js contribute to the maintainability of the system, particularly for individuals well-versed in these technologies.

5) Problem answer interface

Fig. 3 shows the problem answer interface of CPLAS for the code writing problem. To allow students to solve the assigned problem efficiently, the following features are implemented:

- The last answer code submitted by the student is recorded and shown in the code input area to avoid starting all over again.
- The buttons to select the next and previous problems are added.
- The buttons to select the theme that the students prefer.
- Syntax Highlight is included to increase the readability of the code.
- The expected output is updated every time the new input list is generated to allow the students to be able to refer to the expected output for the current input list.
- The output of the students is updated every time the students submit their code.


Offline CPLAS

TOP END

Sidemenu
- Problems
- Submission

Problems

Problem No	ProblemName	Remark
1	max_number	Finished
2	fizzbuzz	Finished
3	fibonacci	Finished
4	pyramid_1	Finished
5	pyramid_2	Finished
6	pyramid_3	Finished
7	pyramid_4	Finished
8	sum_diff	Trying

Fig. 4. Problem list interface of code writing problem.

6) Problem list interface

Fig. 4 shows the list of the problem instances to be solved in this code writing problem category. Here, the problems that have not been solved will be shown as "Trying" and correctly solved problems will be shown as "Completed".

D. Implementation of Code Validation Function

This section presents the method of implementing the code writing problem using Python. The version of Python that is being used is version Python 3.10.6. Fig. 5 shows the overview of the implementation of the code validation function in CPLAS.

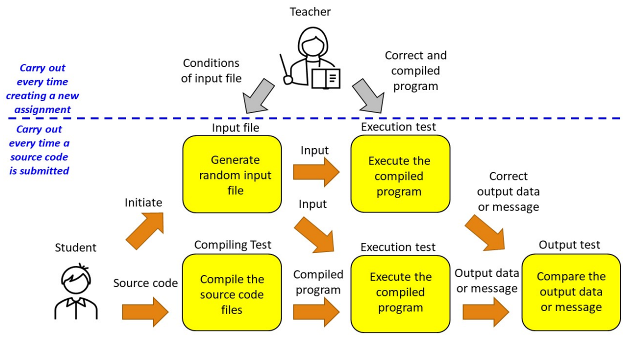

Teacher
Carry out every time creating a new assignment
Conditions of input file
Correct and compiled program
Carry out every time a source code is submitted
Input file
Generate random input file
Input
Execution test
Execute the compiled program
Initiate
Input
Correct output data or message
Student
Compiling Test
Execution test
Output test
Source code
Compile the source code files
Compiled program
Execute the compiled program
Output data or message
Compare the output data or message

Fig. 5. Code validation function procedure.

1) Generate random input file

In the code validation function in CPLAS, the input file prepared by the teacher needs to be random and changes every time the student submits his/her source code. This is because the input list and expected output are shown to the student in the CWP user interface. Students can just use printf to pass the output test. Therefore, a random input file generator that takes the necessary conditions to create the random input file is implemented. Every time a student submits a source code, it will generate an input file according to the conditions specified by the teacher for the specific question.

The random input file generator is built using Python. To make it easy for the teacher to set the conditions of the input file, all the options can be written in the input file name. Then, the input file generator will parse the input file name and create a list of inputs according to the options. Normally, the input file name is written as {input type} {option} inputfile.txt. There are a few types of input types and ranges of input. Input types include int, double/float, and char. All of the available options are shown in Tables I, II, and III. N, R, and D in the table represent a positive number, range expressed in the following way (smaller number – bigger number), and lastly the number of digits and decimals expressed in the following way (number of digit.number of decimal) respectively.

TABLE I. OPTION FOR INT INPUTFILE.TXT

Int Option	Function
_N_ / _N__	N number of positive integer (0–10)
_N_R_ / _N__R_ / _N_R__	N number of positive integer in R range
_N_N1_ / _N_N1__	N number of positive integer and N1 number of negative integer (−10–10)
_N_N1_R_	N number of positive integer and N1 number of negative integer in the R range
__N_ / __N__	N number of negative integer (−10–0)
__N_R_	N number of negative integer in R range

TABLE II. OPTION FOR DOUBLE INPUTFILE.TXT

Double Option	Function
_N_/ _N__	N number of positive number (0–10) with 6 decimals
_N_D_ / _N__D_ / _N_D__	N number of positive number (digits.decimals)
_N_N1_ / _N_N1__	N number of positive integer and N1 number of negative integer (−10–10) with 6 decimal
_N_N1_D_	N number of positive integer and N1 number of negative integer (digits.decimals)
__N_ / __N__	N number of negative integer (−10–0) with 6 decimals
__N_D_	N number of negative integer (digits.decimals)

TABLE III. OPTION FOR CHAR INPUTFILE.TXT

Char Option	Function
_N_ / _N__	N number of character (a–Z)
_N_N1_ / _N_N1__	N number of string with length of N1
__N_ / __N__	N number of word
__N_R_	N number of sentence (R range of sentence length)
__N_D_	N number of sentence (Number of words)
_R_	R range of string (Show number of character)
_D_	D length of sentence (Show number of word)

For example, a file name with **int_3_inputfile.txt** will create an input file with 3 integers. **double__5_4.10_inputfile.txt** will create an input file consisting of 5 double numbers with 4 digits and 10 decimals. **char__6_inputfile.txt** will create an input file consisting of 6 words. In addition, **int_4_1-100_inputfile.txt** shown in Fig. 6 will create an input file consisting of 4 integers within the 1 to 100 range.

2) Correct answer preparation

In the correct answer preparation step for the CWP, the teacher needs to prepare a compiled program of the correct answer code for the assignment. The correct answer code has to be compiled using gcc 11.3.0 in WSL 2 Ubuntu 22.04 distribution. Then, the compiled program is saved as admin exe in the assignment folder.

C programs that are compiled by gcc are targeted to the operating system directly and they cannot be run on other operating systems. However, the adoption of Docker [11] allows the compiled program to run in a Docker containter [12] on all platforms. Thus, building a docker image with Ubuntu base allows the compiled C program to run in a container on all platforms.

Fig. 6 shows the files needed to be prepared by the teacher. These include a template file for the student with assignment name (max_number.c), a compiled program of the correct answer code (admin exe), an output of the correct answer code (admin_output.txt), an input file to generate random inputs for the specific problem (int_4_1-100_inputfile.txt) and the problem statement (problem.txt).

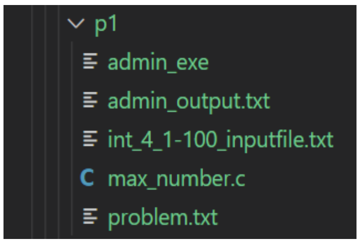
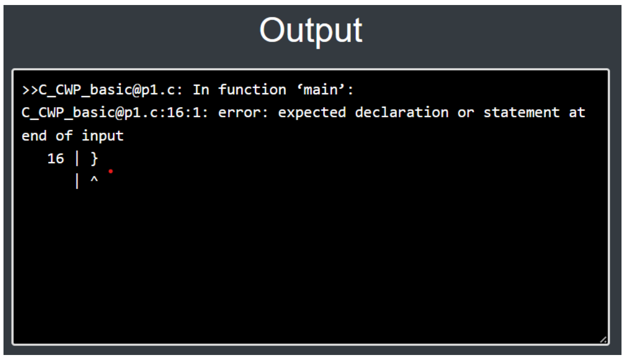

```
p1
  admin_exe
  admin_output.txt
  int_4_1-100_inputfile.txt
  max_number.c
  problem.txt
```

Fig. 6. Files needed to be prepared for one assignment.

3) Compiling test

When a student submits a source code, the code will be saved in submit folder which is inside an addon directory. In the compiling test, the source code files are compiled by using the gcc compiler. An error message will be shown to the student in the output text area if syntax errors are detected by the compiler.

Fig. 7 shows an example of compile error message. The student forgot to put "," at the end of line 10 which cause the compiler to output the error message. Those error messages and output data will be saved in the result folder.



```
>>C_CWP_basic@p1.c: In function ‘main’:
C_CWP_basic@p1.c:16:1: error: expected declaration or statement at
end of input
   16 | }
      | ^
```

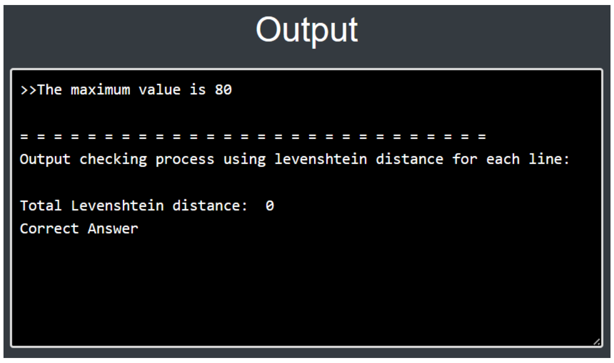
Fig. 7. Example of compile error message.

4) Execution test

In the execution test, the execution code that is obtained in the compiling test is run, with the random input list. Then, if some errors are detected by the Operating System (OS) while running the code, such as the incorrect input data format, the zero division, the buffer overflow, and the infinite loop, this function will report the corresponding message from the OS. For the infinite loop, the maximum running time is set at the execution. If the code passes this test, the next test will be applied.

5) Output test

After the execution test, the output test is performed. This test checks the Levenshtein distance between the output of the student source code with the correct output stored in admin_output.txt. Lavenshtein distance allows the student to know which line of his/her answer code needs to be modified to reach the correct answer. If the distance is 0, the output text will be shown as the "correct answer" in the output text area of the interface. Fig. 8 illustrates the output messages shown in the output text area when the student's answer code is correct. Fig. 9 presents the output messages shown in the output text area when the student's answer code is not correct.

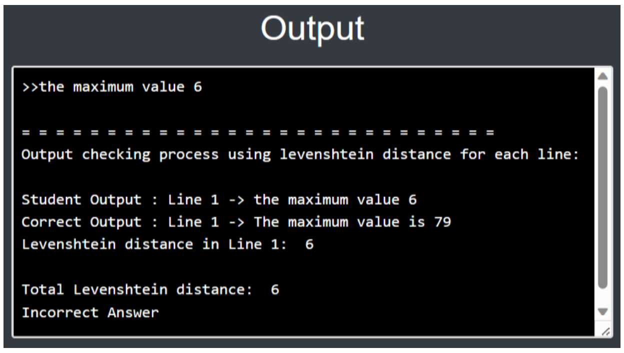


```
>>The maximum value is 80

= = = = = = = = = = = = = = = = = = = = = = = = = =
Output checking process using levenshtein distance for each line:

Total Levenshtein distance:  0
Correct Answer
```

Fig. 8. Example of the correct answer output message.



```
>>the maximum value 6

= = = = = = = = = = = = = = = = = = = = = = = = = =
Output checking process using levenshtein distance for each line:

Student Output : Line 1 -> the maximum value 6
Correct Output : Line 1 -> The maximum value is 79
Levenshtein distance in Line 1:  6

Total Levenshtein distance:  6
Incorrect Answer
```

Fig. 9. Example of the incorrect answer output message.

E. *Adoption of Docker*

In this section, to help students install the CPLAS platform into their computers, we present the adoption of Docker.

1) CPLAS docker workflow

The following procedure describes the steps of adopting Docker for the installation of the CPLAS on a PC.

- Dockerfile Creation: A Dockerfile is a text-based document that outlines the sequence of instructions for copying files and installing the necessary software for the application. In this particular application, Node.js and Python serve as the chosen software platforms. However, due to the usage of some Linux software such as gcc, ubuntu is used as the base image for building the Docker Image through the command shown in Fig. 10. After that, Node.js, gcc, as well as pip (Python package manager), are installed with the code as shown in Fig. 11. The important dependencies of Node.js such as Express.js and EJS are described in package.json, together with packages used in the Python program are included in the Dockerfile through code shown in Fig. 12.

```
FROM ubuntu: latest
```

Fig. 10. Ubuntu as base image.

```
RUN apt-get update; apt-get install -y curl \
        && curl -sL https://deb.nodesource.com/setup_14.x |
        bash - \
        && apt-get install -y nodejs \
        && curl -L https://www.npmjs.com/install.sh | sh
RUN apt-get install -y \
        Python3-pip \
        gcc
```

Fig. 11. Installing required application.

```
RUN pip install –upgrade \
        Lavenshtein \
        func_timeout \
        datetime
RUN npm install
```

Fig. 12. Installing required packages.

- Dockerfile Image Building: In order to create a Docker image, we need to download and install the required platforms and libraries and then package them into a single image, which can be stored on Docker Hub [13] or a local registry. Docker will follow the instructions in the provided Dockerfile in the specified order. After each instruction is executed, the result is cached, and the build proceeds to the next one. Once all the instructions in the Dockerfile have been completed, the Docker image will be constructed on the local machine. After navigating to the application folder which contains the Dockerfile previously, running the code shown in Fig. 13 line 1, the docker image named "cplas" is built. Executing Fig. 13 line 4 allows the generated Docker image to be uploaded to Docker Hub which can be accessed by students.
- Container Running: Generated Docker image can run as the container using Docker "run" command. The command includes the information on port settings as shown in Fig. 14 to reduce the complication for the students.

Then, it is checked that the student answer files are successfully saved in the three folders named "results", "output" and "submits" in the container directory that is mounted with the local file system for accessing by the user.

```
sudo docker build -t cplas
sudo docker push chaixm/cplas:latest
```

Fig. 13. Docker image building.

```
docker run -p 2000:2000/tcp chaixm/cplas
```

Fig. 14. Docker run command.

2) Usage of docker-based CPLAS

The distribution of the CPLAS Docker image from the administrator to the student who will use it can be done through Docker Hub. The CPLAS Docker image can be stored in the Docker Hub account using Docker "push" command as shown in Fig. 13 line 3. Creating a Docker Hub account is a straightforward process on the Docker Hub website, a platform provided and maintained by Docker for discovering and sharing container images. Students with access to the CPLAS platform can retrieve the CPLAS Docker image from Docker Hub using the following command in Fig. 15.

```
docker pull chaixm/cplas:latest
```

Fig. 15. Docker pull command.

3) System installation

To utilize this platform, students should perform the following steps which are outlined in the accompanying manual file:

- Download and install Docker according to the Operating System (OS) of his/her personal computer. For Windows OS, the student is required to install Windows Subsystem for Linux (WSL).
- Use the Docker pull command to download the CPLAS Docker image from Docker, after running Docker on the computer.
- Use the Docker run command to run the image and expose it to the correct port in the computer.
- Open the browser and type localhost:2000 to access the CPLAS platform.

III. Evaluation

In this section, we evaluate the Code Writing Problem (CWP) in CPLAS through applications to nine students at the Tokyo University of Agriculture and Technology and one student at the Niigata University to install and use the implemented CPLAS platform, by referring to the user manual.

A. Solution Result by Students

Table IV shows the number of submissions by each student. Correct answers are colored in green while the wrong answers are colored in red. If a teacher finds that a lot of re-submissions have been done by many students for a particular assignment, this assignment can be considered too difficult such as 7, 9, 10, and 11 for the students. If a teacher finds a student who submitted codes more times than other students, this student needs more attention from the teachers. In this logging system, it is discovered some students' logs are not recorded, thus, this error needs to be fixed in future works.

B. Feedback from Students

Then, we asked the feedback from students with 9 different questions with a score from 1 to 5. We requested 9 students from the Tokyo University of Agriculture and Technology and 1 student from the Niigata University to install and use the implemented CWP on the CPLAS platform, by referring to the user manual. Then, the feedback from the 10 students on 9 questions is recorded. Table V shows the feedback from those students where 5 is the best evaluation and 1 is the least. Most students were satisfied with this platform. Thus, the validity is confirmed. However, on the other hand, a few students discovered some errors in this CPLAS platform that need to be fixed in future works. The errors and their proposed solutions in the future are shown as follows:

- **Problem 1**: When getting into an infinite loop the time limit is too long causing the student to reload the page and lost his/her written answer.
- **Solution 1**: Reduce the time limit and save the written answer before proceeding to other tasks.
- **Problem 2**: Submit button cannot be clicked without scrolling the page.
- **Solution 2**: Fix the Submit button.
- **Problem 3**: Even though the answer is correct the main page still shows that it is in the "Tring" status instead of "Completed" and vice versa.
- **Solution 3**: Fix the JavaScript function to show the correct status of each problem.
- **Problem 4**: It will be better if the number of problems is shown for each assignment and a hint for each question.
- **Solution 4**: Add the number of problems for each assignment and give hint in the template file.
- **Problem 5**: The download button does not function correctly.
- **Solution 5**: Fix the JavaScript function to download the students' answer codes correctly.
- **Problem 6**: Levenshtein distance is not accurate if the mistake is just a line break.
- **Solution 6**: The Lavenshtein distance used in CPLAS is checked line by line. The assignment problems in

  CWP are designed for the students to follow the expected answer output exactly even for line breaks and spaces. Therefore, it will show mistakes even if an extra line break is inserted in the output of the student's answer code.
- **Problem 7**: Several typos are found in the problem statement.
- **Solution 7**: Correct the problem statement.

TABLE IV. STUDENTS' RESULTS AND NUMBER OF SUBMISSIONS

Student/ Question	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
1	🟩 4	🟩 12	🟩 6	🟩 6	🟩 1	🟥 1	🟥 1	🟥 1	🟥 1	🟥 1	🟥 1
2	🟩 1	🟩 3	🟩 2	🟩 2	🟩 1	🟩 9	🟩 5	🟩 7	🟩 8	🟩 3	🟥 1
3	🟩 1	🟩 1	🟩 1	🟩 2	🟩 1	🟩 1	🟩 3	🟩 1	🟩 1	🟩 2	🟥 6
4	🟩 12	🟩 12	🟥 1	🟩 3	🟩 6	🟥 1	🟥 1	🟥 1	🟥 1	🟥 1	🟥 2
5	🟩 2	🟩 1	🟩 4	🟩 3	🟩 3	🟩 2	🟩 8	🟩 3	🟩 1	🟩 21	🟩 2
6	🟩 2	🟩 4	🟩 2	🟩 1	🟩 1	🟩 7	🟩 3	🟩 6	🟩 13	🟩 12	🟩 13
7	🟩 5	🟩 3	🟩 4	🟩 2	🟩 1	🟩 1	🟩 5	🟩 9	🟩 15	🟩 3	🟩 5
8	🟩 3	🟩 3	🟩 5	🟩 5	🟩 1	🟩 1	🟩 11	🟩 13	🟩 17	🟩 34	🟩 42
9	🟩 5	🟩 2	🟩 5	🟩 4	🟩 1	🟩 3	🟩 2	🟩 4	🟩 2	🟩 5	🟩 8
Avg. no. of submission	3.89	4.56	3.33	3.11	1.78	2.89	4.33	5.00	6.56	9.11	8.89
SD of no. of submission	3.21	4.09	1.76	1.52	1.62	2.85	3.16	3.92	6.40	10.78	12.28
Avg. correct rate	1.00	1.00	0.89	1.00	1.00	0.78	0.78	0.78	0.78	0.78	0.56
SD correct rate	0.00	0.00	0.31	0.00	0.00	0.42	0.42	0.42	0.42	0.42	0.50

TABLE V. QUESTIONS AND RESULTS OF FEEDBACK ON CPLAS

No.	**Question**	**Number of students on each score**				
		1	**2**	**3**	**4**	**5**
1	Is it easy for you to solve C programming using CPLAS?	0	2	3	4	1
2	Are the instruction in the manual file of CPLAS clear?	0	0	2	3	5
3	Is the installation process you did for using CPLAS easy for you?	1	2	1	4	2
4	Do you think that the marking is accurate?	0	1	1	3	5
5	Do you feel that showing Levenshtein distance for incorrect problem is motivated for you to solve the problems again?	0	4	2	2	2
6	Do you want to solve the problem which is already corrected?	1	5	4	0	0
7	Do you feel that your programming study is improved after solving the problems in CPLAS?	0	1	2	5	2
8	Are you satisfied using CPLAS in C programming study?	0	0	6	3	1
9	How many rates do you want to give the CPLAS system overall?	0	0	6	4	0

IV. CONCLUSION

This paper presented the Code Writing Problem (CWP) for C Programming Learning Assistant System (CPLAS). For evaluations, 11 CCP instances were generated and assigned to 9 students from the Tokyo University of Agriculture and Technology and 1 student from Niigata University. Their solution results showed that the CWP instances are generally suitable for C programming study by novice students. Then, we asked for feedback from those 10 students with 9 questions. The answer score for the feedback is from 1 to 5, where 5 is the best evaluation and 1 is the least. Overall, we got good evaluation feedback. However, through this application to students, we found some small problems in the implementation. In the future, we will fix those problems and improve the random input generator.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

Htoo Htoo Sandi Kyaw wrote the paper; Chai Xu Min conducted the research and analyzed the data; Keiichi Kaneko advised on the experiments; Soe Thandar Aung helped in system implementation; Nobuo Funabiki advised on the system implementation; Annisa Anggun Puspitasari helped in system implementation; all authors had approved the final version.

ACKNOWLEDGMENT

We are very grateful to our laboratory members for the fruitful discussions of advancing this research. We would also like to thank all the students who participated in the experiments.

REFERENCES

[1] Top programming languages 2022. IEEE Spectrum survey. [Online]. Available: https://spectrum.ieee.org/top-programming-languages-2022

[2] X. Lu, S. T. Aung, H. H. S. Kyaw, *et al.*, "A study of grammar-concept understanding problem for C programming learning," in *Proc. LifeTech*, 2021, pp. 162–165.

[3] X. Lu, N. Funabiki, H. H. S. Kyaw, *et al.*, "Value trace problems for code reading study in C programming," *Adv. Sci. Tech. Eng. Syst. J. (ASTESJ)*, vol. 7, no. 1, pp. 14–26, 2022.

[4] H. H. S. Kyaw, N. Funabiki, S. L. Aung, N. K. Dim, and W.-C. Kao, "A study of element fill-in-blank problems for C programming learning assistant system," *Int. J. Inform. Edu. Tech. (IJIET)*, vol. 11, no. 6, pp. 255–261, 2021.

[5] H. H. S. Kyaw, E. E. Htet, N. Funabiki, *et al.*, "A code completion problem in C programming learning assistant system," in *Proc. ICIET*, 2021, pp. 34–40.

[6] X. Lu, S. Chen, N. Funabiki, M. Kuribayashi, and K. Ueda, "A proposal of phrase fill-in-blank problem for learning recursive function in C programming," in *Proc. IEEE LifeTech*, 2022, pp. 127–128.

[7] A. A. Puspitasari, N. Funabiki, X. Lu, *et al.*, "An implementation of code validation function in C programming learning assistant system," *Int. J. Inform. Edu. Tech. (IJIET)*, vol. 9, no. 1, pp. 24–30, 2023.

[8] D. Herron, *Node.js Web Development*, Packt Publishing, 2016.

[9] Express. [Online]. Available: https://expressjs.com/

[10] HTTP. [Online]. Available: https://nodejs.dev/learn/build-an-http-server

[11] R. McKendrick, *Monitoring Docker*, Packt Publishing Ltd, 2015.

[12] A. Mouat, *Using Docker: Developing and Deploying Software with Containers*, O'Reilly Media, Inc., 2015.

[13] Docker Hub. [Online]. Available: https://hub.docker.com/signup