PAPER

PROVIT-CI: A Classroom-Oriented Educational Program Visualization Tool*

Yu YAN[†a)], Kohei HARA[††], Takenobu KAZUMA[†], Yasuhiro HISADA[†], *Nonmembers*, *and* Aiguo HE[†], *Member*

SUMMARY Studies have shown that program visualization(PV) is effective for student programming exercise or self-study support. However, very few instructors actively use PV tools for programming lectures. This article discussed the impediments the instructors meet during combining PV tools into lecture classrooms and proposed a C programming classroom instruction support tool based on program visualization — PROVIT-CI (PROgram VIsualization Tool for Classroom Instruction). PROVIT-CI has been consecutively and actively used by the instructors in author's university to enhance their lectures since 2015. The evaluation of application results in an introductory C programming course shows that PROVIT-CI is effective and helpful for instructors classroom use.

key words: *program visualization(PV), educational PV tool, introductory programming education, C language, classroom instruction*

1. Introduction

Program Visualization (PV) is useful to illustrate information inside computer programs during their runtime dynamically or statically [1]. It is widely used in visual debugger of Integrated Development Environment(IDE) [2]. On the other hand, studies have shown that PV can also improve beginner's understanding of computer program [3], [4]. A lot of educational PV tools have been proposed to support computer programming learning. For example, Jeliot3, for full or semi-automatic visualization of Java programs [5], [6], has been used in distance education [7], collaborative learning [8] and student programming exercise [9]; VILLE, for visualization of Java and pseudo-code programs, was used in lab session [10]; VIP is a profitable visualization tool for students to learn introductory C++ programming [11]; UUhistle, for introductory Python programming learning, can show program execution proceeds and explore students' mistakes by "visual program simulation exercises"[12], [13]; On-line Python Tutor, a web-based program visualization tool with digital textbooks and example programs works on all modern web browsers for students in exercise sessions or self-study [14]; The Teaching Machine(TM) [15], [16] can visualize C++ and Java program; A useful PV tool was proposed to promote students' understanding of data processing algorithms [17].

In many universities like author's university, a computer programming course consists of lectures and exercises. The exercise is performed in an exercise classroom where each student uses a computer for self-study and the instructor mainly deals with the questions from the students. On the other hand, the lecture is performed in a lecture classroom equipped with an instructor computer and a wall-size projection screen for showing the desktop of the computer. In the lecture classroom, students share the teaching materials including example program source codes by the projection screen and are not requested to use their own computers. However, IDEs and above educational PV tools have only been used to support students in exercise or self-study and there is no research showing that instructors, who play the most important role in programming education, actively integrated those tools into their lecture classroom instruction.

There are impediments that lead to instructors' dissatisfaction when they try bringing those tools into such lecture classrooms:

- Those tools are designed for students to use in an exercise classroom or for self-study. It is difficult for instructors to show the execution of example programs on the projection screen in the lecture classroom by using them. For example, it takes additional time for the instructors to load, type or paste example program source codes into those tools; the layout of their graphical user interface and the visibility of their visualization results are mainly for personal use and not suitable to be shown on the projection screen;
- In some PV tools such as VILLE and the tool of literature [17], the instructors can control and customize the visualization of the target example program for enhancing students'understanding of the program or the algorithm. However, those tools require additional information such as "role information of variables"[18] or "visualization policy" for the program. Therefore, if those tools are applied to the lecture classroom, the following problems will occur: the instructors need to spend additional working time to define those information for the program and it is not easy to show a variation of the program by instantaneously editing it.

In order to apply educational PV tools to enhancing instructions in the lecture classrooms, it is necessary to improve those tools being **classroom-oriented**. Here, **classroom** means the **lecture classroom** described above and

Manuscript received April 20, 2017.
Manuscript revised August 30, 2017.
Manuscript publicized November 1, 2017.

[†]The authors are with Graduate School of Computer Science and Engineering, The University of Aizu, Aizu-wakamatsu-shi, 965–0006 Japan.
[††]The author is with Maple Systems Co., Ltd., Tokyo, 104–0061 Japan.

[*]This is a paper on system development.

a) E-mail: d8171106@u-aizu.ac.jp

DOI: 10.1587/transinf.2017EDK0002

classroom-oriented PV tool should have following features:

- It is easy for instructor to use in the classroom;
- It only adds minimum work to instructor to use it;
- It does not show any unnecessary information on the projection screen in the classroom;
- It offers visualization results with high enough visibility in the classroom to students;
- It does not require any change in the appearance of original example program source codes and teaching materials.

This article proposes a classroom-oriented PV tool–PROVIT-CI(PROVIT for Classroom Instruction). The core of it is PROVIT(PROgram VIsulization Tool), an educational PV tool developed in author's university [20]. PROVIT-CI has been experimentally used in a regular introductory C programming course in author's university and its usefulness has been confirmed by the experimental application.

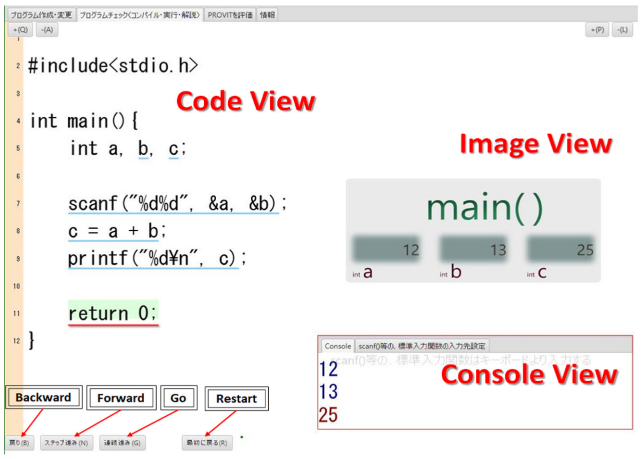
The remainder of the article is structured as follows: Sect. 2 reviews PROVIT; Sect. 3 describes PROVIT-CI in details; Sect. 4 states an implementation of PROVIT-CI; Sect. 5 describes the experimental application of PROVIT-CI and discusses the evaluation results.

2. PROVIT

PROVIT is aimed at visualizing the processing of C program execution to improve beginner's understanding. It is developed by Java technology, thus, supported by multiple kinds of operating system. It includes a C virtual machine [21], a program code editor and Run Viewer for user to write, run and check their C programs. Figure 1 shows Run Viewer of PROVIT. It has a Code View, an Image View and a Console View.

Run Viewer offers following basic functions for the user to check his program:

- One step execution by clicking "Forward" button;
- Back to previous step by clicking "Backward" button;



```
#include<stdio.h>

int main(){
    int a, b, c;

    scanf("%d%d", &a, &b);
    c = a + b;
    printf("%d¥n", c);

    return 0;
}
```

```
12
13
25
```

Fig. 1 Run Viewer of PROVIT

- Continuative execution to breakpoint or the end of the program by clicking "Go" button;
- Execution from the beginning of the program by clicking "Restart" button.

Following above functions, Code View displays the source code of the program as follows:

- Indicate all steps that have been executed with light blue color underlines for the user to trace the runtime execution of the program;
- Indicate the step that will be executed in next one step execution with a red color underline;
- Set or clear a breakpoint by clicking on any step of the source code and the breakpoints are indicated by light green rectangles.

Image View visualizes current status of the running C program by graphically showing following information:

- Variable data type, variable name and its current value;
- Function name.

Console View simulates the console of the computer, showing standard input and output of the program.

Although PROVIT has many useful visualization functionalities, we found the following problems when PROVIT was used in a regular programming course for the first year students in author's university:

- In this course, basic concepts about array data structure such as dimension, size and zero-based indexing are taught and then, students learn applications of large-size 2-dimensional array. However, many students had difficulty understanding them.
- In many example programs used in this course, return values are directly referenced in expressions like:

```
if(scanf("%d", &a) != 1){
    (do something)
}
```

However, in order to show a return value by using PROVIT, the source code needs additional variable and assignment statement such as:

```
int v = scanf("%d", &a);
if(v != 1){
    (do something)
}
```

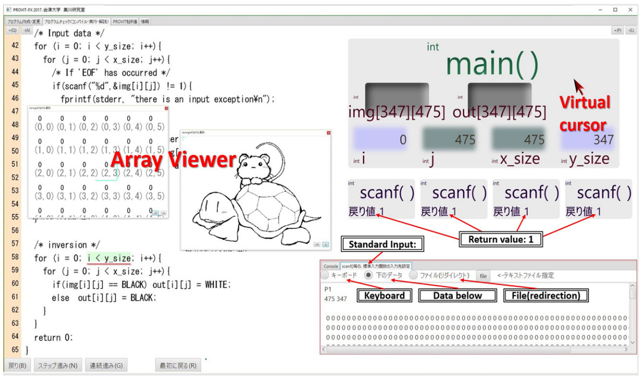
Therefore, the instructors have to change those example programs when they want to use PROVIT.

3. PROVIT for Classroom Instruction (PROVIT-CI)

PROVIT-CI is an extension of PROVIT for instructors to use in the lecture classroom described in Sect. 1 for teaching basic programming concepts and techniques. In order to solve the problems of PROVIT mentioned in Sect. 2 and promote students' understanding during the lecture, PROVIT-CI has been enhanced as follows:

- In Run Viewer, a big virtual cursor is added to replace the usual mouse cursor for student to easily trace the elements which are being instructed on the views;
- In Code View, the red color underline is replaced by a blinking red color underline;
- In Image View, variable's role is shown by different color: variable(s) which will be referenced at next one step execution are marked by blue color background and variable which will be changed at next one step execution is marked by red color background. Figure 2 shows an example of variable visualization. Here, step "i < y_size" in line 58 of the program will be executed at next one step execution and variables "i" and "y_size" with blue color background will be referenced in that execution;
- Information of completely executed functions is displayed in Image View if the return value of the function will be referenced by other step. Figure 2 also shows an example of function visualization. Here, "scanf()" was repeatedly called at line 45 of the program and its return value was referenced. The last 4 results of function call including the type of the return value were displayed;
- Array Viewer for visualization of array variable was added. It can display the value in each element of the array or show all the values in the array as a monochrome image since in introductory programming course array is usually used to process simple image data. As shown in Fig. 2, the example program which is taught to the first year students in author's university loads a monochrome image into the first array and saves its inversion to the second array. Here, the left Array Viewer shows the values in part elements of the first array and the right one shows the whole data in the second array as a monochrome image;
- All of three views of Run Viewer support zoom in or zoom out operation to show the details of visualization elements.

In order to save class time, PROIVT-CI is also designed to provide convenient and easy operation for the lecture classroom use, as follows:



```
   /* Input data */
42   for (i = 0; i < y_size; i++) {
43     for (j = 0; j < x_size; j++) {
44       /* If 'EOF' has occurred */
45       if(scanf("%d",&img[i][j]) != 1) {
46         fprintf(stderr, "there is an input exception¥n");
47
48
49
50
51
52
53
54
55
56
57   /* inversion */
58   for (i = 0; i < y_size; i++) {
59     for (j = 0; j < x_size; j++) {
60       if(img[i][j] == BLACK) out[i][j] = WHITE;
61       else  out[i][j] = BLACK;
62     }
63   }
64   return 0;
65 }
```

```
  0      0      0      0      0      0
(0, 0) (0, 1) (0, 2) (0, 3) (0, 4) (0, 5)
  0      0      0      0      0      0
(1, 0) (1, 1) (1, 2) (1, 3) (1, 4) (1, 5)
  0      0      0      0      0      0
(2, 0) (2, 1) (2, 2) (2, 3) (2, 4) (2, 5)
  0      0      0      0      0      0
(3, 0) (3, 1) (3, 2) (3, 3) (3, 4) (3, 5)
  0      0      0      0      0      0
```

```
int img[347][475]   int out[347][475]
int i 0   int j 475   int x_size 475   int y_size 347
int scanf( ) 戻り値 1   int scanf( ) 戻り値 1   int scanf( ) 戻り値 1   int scanf( ) 戻り値 1
```

```
P1
475 347

0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
```

Fig. 2 Run Viewer of PROVIT-CI

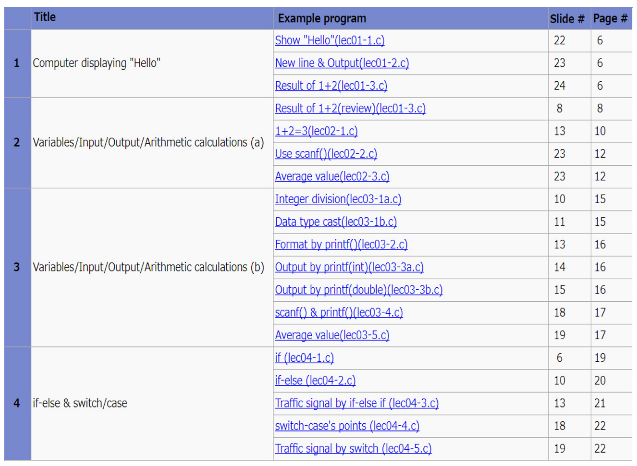
- PROVIT-CI supports standard input redirection. The instructor can set text data or a path of a text file on Console View for the example program being visualized by PROVIT. Figure 2 shows an example of using standard input redirection where the program loads image data by standard input from a text formate file instead of keyboard and the content of the file is also shown in Console View;
- PROVIT-CI offers two kinds of web service to the instructors: (1) Instructor can upload his example program source codes and input data that they want to use in the classroom to the web server of PROVIT-CI before the lecture; (2) For each example program, PROVIT-CI gives an PROVIT URL. When instructor accesses to the URL from a web browser, PROVIT will be launched directly showing its Run Viewer with the example program in its Code View and the input data in its Console View. The URL can be embedded in PowerPoint slide or PDF file of teaching materials. Therefore, instructor can quickly start PROVIT by the URL embedded in on showing material.
- All operations of Run Viewer can be done not only by pointing devices but also keyboard accelerators.

4. Implementation

PROVIT-CI is composed of a web server and PROVIT URLs as client interface. On the client side, PROVIT is launched by Java Web Start technology [22]. In the web server, the body of PROVIT, C example programs, input data needed for the C programs to read by standard input function are placed. Each PROVIT URL corresponds to a HTML file. An example of the HTML file is showed as Listing 1.

Listing 1 Code example of PROVIT URL

```
<html><head>
<script src="./web-files/dtjava.js">
</script><script>
function launchPROVIT(jnlpfile){
  dtjava.launch({
    url : './provit.jnlp',
    params: {
      startMode: "run",
      windowMode: "modal",
      sourceCode: "programFileName",
      inputData: "inputDataFileName"
    },
    jnlp_content : ''
    },
    {
        javafx : '8.0+'
    }
  );
  return false;
}
dtjava.addOnloadCallback(launchPROVIT);
</script>
</head></html>
```



	Title	Example program	Slide #	Page #
1	Computer displaying "Hello"	Show "Hello"(lec01-1.c)	22	6
		New line & Output(lec01-2.c)	23	6
		Result of 1+2(lec01-3.c)	24	6
2	Variables/Input/Output/Arithmetic calculations (a)	Result of 1+2(review)(lec01-3.c)	8	8
		1+2=3(lec02-1.c)	13	10
		Use scanf()(lec02-2.c)	23	12
		Average value(lec02-3.c)	23	12
3	Variables/Input/Output/Arithmetic calculations (b)	Integer division(lec03-1a.c)	10	15
		Data type cast(lec03-1b.c)	11	15
		Format by printf()(lec03-2.c)	13	16
		Output by printf(int)(lec03-3a.c)	14	16
		Output by printf(double)(lec03-3b.c)	15	16
		scanf() & printf()(lec03-4.c)	18	17
		Average value(lec03-5.c)	19	17
4	if-else & switch/case	if (lec04-1.c)	6	19
		if-else (lec04-2.c)	10	20
		Traffic signal by if-else if (lec04-3.c)	13	21
		switch-case's points (lec04-4.c)	18	22
		Traffic signal by switch (lec04-5.c)	19	22

Fig. 3 A list of PROVIT URL [24]

The meanings of key elements in the HTML file are as follows:

- "./web-files/dtjava.js" and "./provit.jnlp" are automatically generated by NetBeans [23] to start PROVIT as a JavaFX application;
- Parameter "startMode" with value "run" controls PROVIT to directly load the example program and the input data showing them by Run Viewer;
- Parameter "windowMode" with value "modal" controls PROVIT to show its window on the top of the client computer screen and ready to be used by the instructor;
- "*programFileName*" indicates the source code file path name of the example program;
- "*inputDataFileName*" indicates the path name of text data file for the program to read as standard input.

Figure 3 shows an example of PROVIT URL list loaded in a web browser. This list includes the PROVIT URLs for all example programs used for a first-year programming course in author's university and for each program, there is following information:

- Lecture Number;
- Lecture Description;
- Title of example program embedded with a hyperlink to its PROVIT URL;
- Teaching material information corresponded to the program.

5. Evaluation on Experimental Application

Since 2015, as an experimental application, PROVIT-CI has been introduced to the classroom of a regular course, "Introduction of Programming" in author's university. This course is given to 240 first-year students by 3 instructors from April to July for 16 weeks every year. Each week has a 90 minutes lecture teaching C language and computer programming. Table 1 shows the main topics and basic concepts taught in each week of the course. The teaching materials are made by Microsoft PowerPoint.

Table 1 Course distribution

Lec.#	Topics and Concepts
1	Introduction, "Hello world"
2	Variable, I/O, Assignment, Arithmetic operations(a)
3	Variable, I/O, Assignment, Arithmetic operations(b)
4	If statement & Switch-case statement
5	While statement
6	For statement
7	1-Dimensional Array
8	Special Lecture by invited instructor
9	Midterm Exam
10	Program structure & Flowchart
11	2-Dimensional Array
12	Input redirection & Pipe; Image-Processing
13	Function(a)
14	Function(b)
15	Coding style
16	Final Exam

Before the experiment, the following preparation was done:

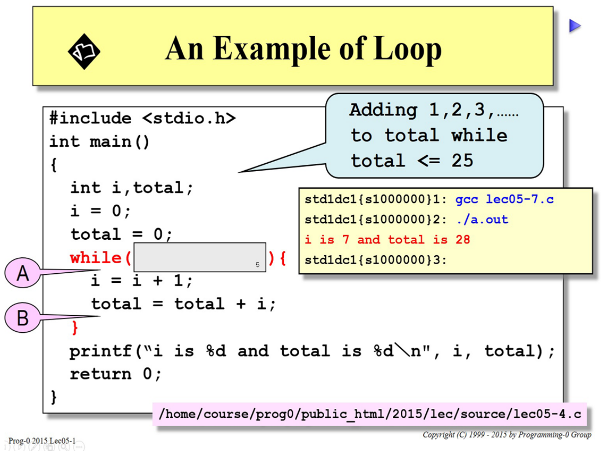
- All example program source codes that would be used by the instructors in their lectures and input data for the standard input of example programs were uploaded on PROVIT-CI server;
- For each slide including an example program, a hyper link of PROVIT URL corresponding to the program was embedded for instructors to easily run the program by PROVIT;
- Since the user interface of PROVIT is very simple, there is no any instruction document about it being offered to the instructors. Only a short explanatory meeting about PROVIT's operation was held for them.

The instructors were not forced to use PROVIT-CI. Therefore, during the lectures, the instructors could show their PowerPoint slides and explain the example program source code on the slide as usual. When the instructor wanted to run the program instantly and trace its execution step by step, he only needed to click the hyper link embedded on the slide to start PROVIT-CI which shows a full screen window that overlays on the slide. After the explanation was finished, the instructor could close PROVIT-CI and show the original slide again.

Figure 4 shows an example of teaching material by which the instructor showed the source code, compile command and execution result of an example program. Figure 5 shows the sight of a classroom where the instructor had launched PROVIT-CI to explain the runtime behavior of the same example program in Fig. 4. Unlike existing IDEs and educational PV tools, PROVIT-CI can display the source code and visualization results by high enough visibility to the students.

During the experiment in 2015 and 2016, PROVIT-CI was not used in several lectures because of following reasons:

- Lecture 8,9,10,15 and 16 were not for teaching new



An Example of Loop

```
#include <stdio.h>
int main()
{
  int i,total;
  i = 0;
  total = 0;
  while(                 ){
    i = i + 1;
    total = total + i;
  }
  printf("i is %d and total is %d\n", i, total);
  return 0;
}
```

Adding 1,2,3,...... to total while total <= 25

```
std1dc1{s1000000}1: gcc lec05-7.c
std1dc1{s1000000}2: ./a.out
i is 7 and total is 28
std1dc1{s1000000}3:
```

/home/course/prog0/public_html/2015/lec/source/lec05-4.c

A
B
5
Prog-0 2015 Lec05-1
Copyright (C) 1999 - 2015 by Programming-0 Group

Fig. 4 Example of teaching material(PowerPoint slide)

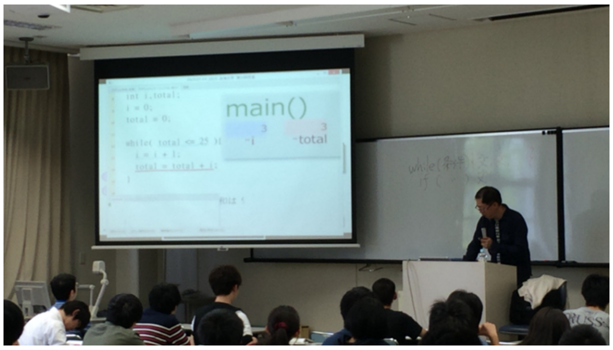


Fig. 5 PROVIT-CI used in a lecture classroom

contents that need to be explained by example programs;

- Some instructors could not access PROVIT server due to network problems in their computer used in the lectures;
- There is a preparatory work error of embedding PROVIT URL into PowerPoint slides.

Instructor's operations of PROVIT-CI were recorded as log data during the experimental application and an evaluation from the students about the effectiveness of PROVIT-CI by a questionnaire was encouraged in 2016.

5.1 Functions Used in Each Lecture

Table 2 is one analysis result of instructor's operation log data: the situation that the instructors used PROVIT-CI's functions for teaching each lecture in 2015 and 2016. The topics taught in each lecture are shown in Table 1 and the functions are listed in Table 3. Here, the value in each cell is the total number of times that the corresponding function has been used at the corresponding lecture in 2015 and 2016.

From Table 2, the following turned out:

- The mostly used function was "To next step" for tracing program execution. "Go", "Restart" and "Set/Clear break point" were also frequently used. Especially "Go" was more frequently used in lecture 6 and lecture 12 for teaching loop statements and applications of large size array. Those functions can shorten the time in explaining example programs;
- "Array Viewer" and "Standard input redirection" were actively used in particular lectures for showing the details and the overview of array variables and automatically reading data from standard input to an array variable. Especially in Lecture 12 that all example programs use large size array variables, it is difficult for the instructors to run those programs and show the results effectively during the lecture without PROVIT-CI;
- "Source code edit" was often used in the lecture classrooms. This function is important in lecture classrooms for the instructors to easily and instantly show a variation of example program or make a correction of the program;
- "Back to previous step" was expected to be used to repeatedly and easily show the change in variable's value caused by the execution of program. However, it was used in only two lectures. That means this function is not necessary in the classroom.

Table 2 Frequency of use of PROVIT-CI's function

Lec.#	Functions								
	1	2	3	4	5	6	7	8	9
1									
2	84	2	1	1				7	
3	81			1				6	
4	5								
5	168		2		2			4	
6	467	2	24	4	5			3	
7	562		4	3	2	19	6	8	1
11	831		2	4	1	25	8	2	
12	307		78	1	8	21	8		
13	155			1	6				
14	166			1	3				

Table 3 Main Functions of PROVIT-CI

No.	Function
1	To next step(one step execution)
2	Back to previous step
3	Go(Continuative execution to breakpoint or the end of program)
4	Restart(Execution from the beginning of the program)
5	Set/Clear break point
6	Array Viewer
7	Standard input redirection
8	Source code edit
9	Source code file load

5.2 Accumulated Use Time

Table 4 and Table 5 show another analysis result of instructor's operation log data: the time that instructors used PROVIT-CI for each lecture in 2015 and 2016. Here, time of "During Lecture" is the total time the instructors used PROVIT-CI during a lecture; time of "Before Lecture" is the total time the instructors used PROVIT-CI for checking example programs, getting familiar with PROVIT-CI on the day before the day when the corresponding lecture was performed. The used time was counted only when PROVIT-CI's window was on the top of instructor's computer display. In each cell of the tables, the total time in minutes that PROVIT-CI was used and the number of instructor who used PROVIT-CI for the corresponding lecture are shown.

From Table 4 and Table 5, the following can be understood.

- PROVIT-CI was actively used in all the lectures in which the instructors were able to use it. The first several lectures were mainly for introducing basic concepts and only one short example program was explained by using PROVIT-CI in each lecture. For showing the execution of a short program by PROVIT-CI, 1 or 2 minutes was enough. In the other lectures, complicated programs were introduced and repeatedly executed thus PROVIT-CI was used longer in each lecture;
- The time of "Before Lecture" can be thought as the only additional burden for the instructors to use PROVIT-CI. This time was not so much, the average of it was only 8.8 minutes. Moreover, in 2016, the time spent before each lecture had been completely decreased.

5.3 Evaluation by Students

To confirm the usefulness of PROVIT-CI, a questionnaire

Table 4 Accumulated use time(in minutes) in 2015

Lec.#	Use time/Number of instructors	
	During Lecture	Before Lecture
1		
2	8.2/1	7.8/1
3	6.7/1	4.0/1
4		12.9/1
5	16.3/2	33.1/1
6	13.3/2	16.5/2
7	15.4/2	15.1/2
11	14.0/1	9.6/2
12	15.1/2	14.7/2
13	1.8/1	9.0/1
14	9.1/2	14.5/2

Table 5 Accumulated time(in minutes) in 2016

Lec.#	Use time/Number of instructors	
	During Lecture	Before Lecture
1		5.7/2
2	2.3/1	2.8/2
3	1.7/1	
4	1.9/1	4.9/1
5		2.7/1
6	27.9/3	5.9/1
7	19.0/2	4.2/1
11	19.9/2	1.5/1
12	8.3/2	1.0/1
13	12.3/2	1.0/1
14		

asking whether PROVIT-CI has been or will be helpful for understanding the example programs showed in each lecture was given to all the students at the last lecture of the second year(2016). Figure 6 presents the statistical results of the answer. The numbers on the horizontal axis correspond to the lecture number in Table 1. The answers according to lecture 2 and 3 are summarized together since the two lectures have same topics and concepts. The same way for lecture 13 and 14. "A" means that the answerer thought PROVIT-CI was or would be very helpful and "E" means that the answerer thought PROVIT-CI was not or would not be helpful at all.

From Fig. 6, the followings can be understood:

- PROVIT-CI was highly accepted by the students. For most of the lectures, more than 75% of students answered that PROVIT-CI is helpful for improving the understanding of the lecture;
- PROVIT-CI solved the problems of PROVIT described in Sect. 2 by introducing new Array Viewer and visualization of return value. Array Viewer can automatically deal with the number of dimensions and the size of the array. And it can easily show the relations between the whole array and any particular element of the array by simple operations. Visualization of return values helps the instructors to easily explain how to simply make use of the return values without using additional variables and assignment statements. Figure 6 shows that PROVIT-CI was very useful for teaching array (in lecture 7 and 11) and function (in lecture 13 and 14). Thus, these new functionalities contribute to promoting students' understanding in lecture classrooms;
- In lecture 1 and lecture 5, PROVIT-CI was not used. However, there were more than 65% of students thought it could be helpful in these lectures if it was used;
- PROVIT-CI used in lecture 12 got a lower evaluation than in other lectures, less than 65% of students thought it is helpful. In this lecture, all example programs were, as applications of array variable, implementations of

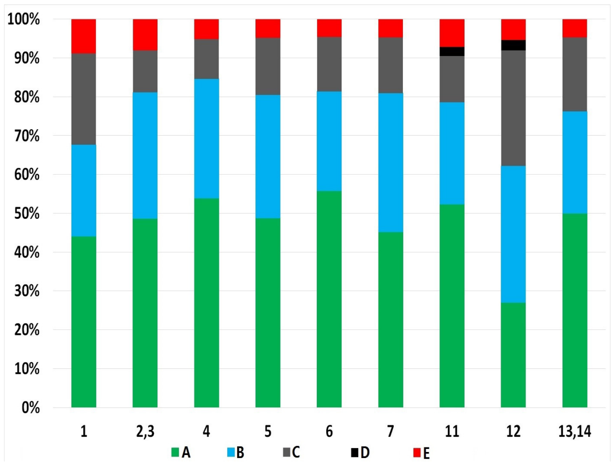


Fig. 6 Evaluation result of PROVIT-CI by students

monochrome image processing and PROVIT-CI's Array Viewer played an important role for the explanation of those programs. The reason of the lower evaluation is that the content of this lecture, based on the curriculum, was originally designed without consideration of using PROVIT-CI and the instructors, for the first time, need to spend a lot of time to teach how to feed image data to the programs by standard input redirection and feed the output of the programs to an image viewer tool by output redirection. And PROVIT-CI is not for teaching those techniques.

5.4 Evaluation by Instructors

Instructors of this course confirmed that PROVIT-CI is effective to promote student's understanding of the example programs shown at lectures. According to the feedback from the instructors, the following features of PROVIT-CI were highly evaluated:

- Quickly start by PROVIT URL;
- Big virtual cursor;
- Operating by keyboard accelerators;
- Easy standard input redirection;
- Array Viewer.

The instructors also confirmed that PROVIT-CI requires no additional information for the visualization of the example programs.

Furthermore, the instructors declared that they will continue to use PROVIT-CI in this course in the future and that they hoped PROVIT-CI can be used in further courses teaching more concepts such as pointer and structure.

5.5 Limitations and Ongoing Work

Although PROVIT-CI was not used in some lectures by any instructor as shown in Table 5, Fig. 6 shows that students considered that PROVIT-CI is useful for all lectures. Therefore, it is necessary to consider using PV tools for more lectures in classroom.

As shown in Table 4 and Table 5, PROVIT-CI could not be used in not a few lectures. The main reason was that the instructors could not connect their mobile computer to the internet at classroom. Therefore, as a classroom-oriented PV tool, only offering on-line service is not enough, service for off-line use is also necessary.

At present, we are planning to add custom-made visualizations for more C language concepts such as pointer, structure so that PROVIT-CI can be useful in more advanced C courses. We are also considering services for the instructors to use PROVIT-CI at lectures without network environment and for the instructors out of author's university to use PROVIT-CI for their own lectures.

6. Conclusion

This study proposes a classroom-oriented educational program visualization tool, PROVIT-CI, with several delicate designs in its GUI and function for improving student understanding, offering more convenience and saving class time. Three instructors in author's university have actively used PROVIT-CI to help them teach programming in the lecture classrooms for two years. The results of the experimental application show that PROVIT-CI can effectively solve the problems in trying to use existing PV tools to help lecture classroom instruction. Though PROVIT-CI is only for teaching programming by C language, the approach proposed and the results achieved in this study are applicable to other computer languages.

Acknowledgments

We would like to thank Prof. Kohei Otsuyama and Prof. Hirohide Demura for applying PROVIT-CI to their lecture classrooms and giving us valuable comments. We would also like to thank Prof. Masayuki Tanimoto for giving us very valuable advice on the preparation of this article.

References

[1] B.A. Myer, "Visual programming, programming by example, and program visualization: a taxonomy," Proc. ACM SIGCHI Bulletin, vol.17, no.4, pp.59–66, April 1986.
[2] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson, "A survey of literature on the teaching of introductory programming," Proc. ACM SIGCSE Bulletin, vol.39, no.4, pp.204–223, Dec. 2007.
[3] G. Ebel and M. Ben-Ari, "Affective effects of program visualization," Proc. 2nd International Workshop on Computing Education Research, Canterbury, United Kingdom, pp.1–5, Sept. 2006.
[4] M.J. Laakso, T. Rajala, E. Kaila, and T. Salakoski, "The impact of prior experience in using a visualization tool on learning to program," Proc. IADIS International Conf. on Cognition and Exploratory Learning in Digital Age, Freiburg, Germany, pp.13–15, Oct. 2008.
[5] A. Moreno, N. Myller, E. Sutinen, and M. Ben-Ari, "Visualizing programs with Jeliot 3." Proc. Working Conf. on Advanced Visual Interfaces, pp.373–376, Gallipoli, Italy, May 2004.
[6] N. Myller, R. Bednarik, and A. Moreno, "Integrating dynamic program visualization into BlueJ: The Jeliot 3 extension," Proc. 7th IEEE International Conf. on Advanced Learning Technologies, Sendai, Japan, pp.505–506, July 2007.
[7] O. Kannusmäki, A. Moreno, N. Myller, and E. Sutinen, "What a novice wants: Students using program visualization in distance programming course," Proc. 3rd Program Visualization Workshop, The University of Warwick, UK, pp.126–133, July 2004.
[8] N. Myller, R. Bednarik, E. Sutinen, and M. Ben-Ari, "Extending the engagement taxonomy: Software visualization and collaborative learning," Proc. ACM Trans. Computing Education, vol.9, no.1, March 2009.
[9] A. Pears and M. Rogalli, "mJeliot: a tool for enhanced interactivity in programming instruction," Proc. 11th Koli Calling International Conf. on Computing Education Research, TBA, Finland, pp.16–22, Nov. 2011.
[10] E. Kaila, T. Rajala, M.J. Laakso, and T. Salakoski, "Effects of course-long use of a program visualization tool," Proc. 12th Australasian Conf. on Computing Education, Brisbane, Australia, pp.97–106, Jan. 2010.
[11] A.T. Virtanen, E. Lahtinen, and H.M. Järvinen, "VIP, a visual interpreter for learning introductory programming with C++," Proc. 5th

Koli Calling Conf. on Computer Science Education, pp.125–130, Nov. 2005.

[12] J. Sorva and T. Sirkiä, "UUhistle: a software tool for visual program simulation," Proc. 10th Koli Calling International Conf. on Computing Education Research, Koli, Finland, pp.49–54, Oct. 2010.

[13] T. Sirkiš, J. Sorva, "Exploring programming misconceptions: an analysis of student mistakes in visual program simulation exercises," Proc. 12th Koli Calling International Conf. on Computing Education Research, Koli, Finland, pp.19–28, Nov. 2012.

[14] P.J. Guo, "Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education," Proc. 44th ACM technical Symposium on Computer science education, Denver, CO, USA, pp.579–584, March 2013.

[15] M.P. Bruce-Lockhart and T.S. Norvell, "Lifting the hood of the computer: Program animation with the teaching machine," Proc. IEEE Canadian Conf. on Electrical and Computer Engineering, Canadian, pp.831–835, May 2000.

[16] M.P. Bruce-Lockhart and T.S. Norvell, "Developing mental models of computer programming interactively via the web," Proc. 37th Annual Frontiers In Education Conf., pp.S3H-3, Jan. 2007.

[17] K. Yamashita, R. Fujioka, S. Kogure, Y. Noguchi, T. Konishi, and Y. Itoh, "Practices of algorithm education based on discovery learning using a program visualization system," Proc. Research and Practice in Technology Enhanced Learning, vol.11, no.4, pp.15, Aug. 2016.

[18] T. Rajala, M.J. Laakso, E. Kaila, and T. Salakoski, "VILLE: a language-independent program visualization tool," Proc. 7th Baltic Sea Conf. on Computing Education Research, pp.151–159, Koli National Park, Finland, Nov. 2007.

[19] T. Rajala, M.-J. Laakso, E. Kaila, and T. Salakoski, "Effectiveness of Program Visualization: A Case Study with the ViLLE Tool," Proc. J. Information Technology Education, vol.7, pp.15–32, 2008.

[20] Y. Yan, H. Hiroto, H. Kohei, S. Shota, and A. He, "A C Programming Learning Support System and Its Subjective Assessment," Proc. IEEE International Conference on Computer and Information Technology (CIT), pp.561–566, Xi'an, China, Sept. 2014.

[21] A. He, "C Virtual Machine for Educational Program Visualization for Beginners," Proc. IEICE Trans. Inf. & Syst., vol.J98-D, no.10, pp.1292–1300, Oct. 2015.

[22] J. Zukowski, "Deploying software with jnlp and java web start," Technical Article, http://www.oracle.com/technetwork/articles/javase/index-135962.html, accessed Aug. 2002.

[23] T. Boudreau, J. Glick, S. Greene, V. Spurlin, and J. Woehr, NetBeans: The Definitive Guide: Developing, Debugging, and Deploying Java Code, O'Reilly Media, Inc, 2002.

[24] http://cleast.u-aizu.ac.jp

Yu Yan completed her Bachelor of Science in Computer Science and Technology from Harbin Normal University, Heilongjiang, China, in 2013 and Masters of Science in Computer Science and Engineering from the University of Aizu, Fukushima, Japan, in 2015. She is currently a Ph.D student in Graduate Department of Computer and Information System, the University of Aizu. Her research interests include Program Visualization (PV), Computer Network, Computer Education and Personalized e-Learning.

Kohei Hara completed his Bachelor of Science in Computer Science and Engineering from University of Aizu, Fukushima, Japan, in 2015 and Masters of Science in Computer Science and Engineering from the University of Aizu, Fukushima, Japan in 2017. He is an employee of Maple Systems Co., Ltd. Tokyo, Japan. His research interests include C language programming exercise education for teachers and students.

Takenobu Kazuma received the Bachelor's degree in 2010 from the University of Aizu, Fukushima, Japan and the Master's degree in Computer Science and Engineering from the University of Aizu in 2012. Now he is a Ph.D student in the University of Aizu. His research interests include Computer Network, Computer Education, Shape Recognition, Human-Computer Interaction, Natural User Interface.

Yasuhiro Hisada received his PhD in Systems and Information Engineering from Toyohashi University of Technology in 1993. Since 1993, he has been with the University of Aizu. He is currently an associate professor at Division of Information Systems, the University of Aizu. His research interests are in biomedical engineering and remote sensing.

Aiguo He now is an associate professor of computer science at the University of Aizu, Fukushima, Japan. His research interests include e-Learning, distributed control system, computer network and software engineering. He obtained his PhD at Nagoya University in 1988 and was working in Intelligent Vision & Image Systems Corporation until 2000. He is a member of IEEE CS, IEEJ, IEICE and IPSJ.