1. Introduction

Programming in computer languages is an important class for university students in courses of computer science, electronics and information engineering. Moreover, there exists a programming class for even all students of a university. We have a programming class in C language, one of most popular computer languages, with a corresponding programming practice for our students who are first year of the university. Most of the students have not had experiences of programming before the practice, nor knowledge about programming on a computer.

1.1 Class and textbooks

However, in the class, the way to teach a computer language is not easy for them to understand because it greatly depends on the contents of a textbook used in the class, which is written for an expert in programming. There are a large number of such books on C such as [1] [2], however, goes with the same way as follows; firstly introducing a program or C, then showing a simple sample program, explanation on data types, loops, arrays, functions and pointers. Books on C programming usually assumes that "the readers have no previous programming experience" [1]. However, what the books describe in the first and second chapter violates the assumption. Tutorials for beginners are usually very small part of if, and the rest is used by details of the language. For example, Fig.1 is first shown to the reader, but the details are given in the latter half of the book. Therefore, the class also spends less time for beginners.

The small part of tutorials at the top of textbooks on computer languages are not basics nor fundamentals to understand computer programming, but just an introduction of a certain programming language. For instance, a first sample code in C language (shown in Fig.1) usually comprises a lot of special terms and concepts; main function, #include directive, printf function and return statement. Although the code is shown in first page of a textbook or explained at top of the first week of the class, even such small code requires initially a good understanding of programming on a computer ("what is function?" or "how uses braces, brackets, semi-colons?"). Furthermore, the terms and concepts are not common to other procedural programming languages; for example, Fig.2 shows that the same program is written in completely different style. These difference between programming languages totally confuse beginners.

Besides, books on programming are not suit for a class. To explain the mechanism of C
in the books, they require the reader to setup the environment of computer so that the reader can input by a text editor sample source codes shown in the books, and compile the sources to make object files, link them to make an executable binary, and execute the binary file to see the result of the source code. The procedure described here needs many operations with keyboard and mouse, in which actual computer experience is needed.

1.2 Practice

On the other hand, in the programming practice that accompanies with the class, programming tasks are set to make actual program codes by using a computer, while students are taught using only textbooks in the class. However, since the class has not yet proceeded to topics suitable for a specific task, the students are confused by the lack of elementary knowledge about programming when they try to make an actual program code.

At the first in the practice, they have not yet had skills of programming and making algorithm because the class has not much proceeded. For example, students have to understand the overview of C programming in the first couple of weeks in the class, however, they have to make a program using a computer in the practice in the first week.

Moreover, because of a lack of even simple knowledge about the syntax in C, a lot of errors occur when a beginner is asked to input source codes. They do not know the difference among braces, brackets, and angle brackets, the meaning of semicolons, the naming of variables and functions. Therefore, they are confused when the computer displays unfamiliar messages such as syntax error in line 10 or Segmentation fault. They have to make out at least the (simple) syntax and the concept of computation in C before they try to make a program in order to avoid trivial errors in the practice.

1.3 Our approach

In this paper, we show our activity to inculcate very basics of programming to students. We have created nearly 250 small exercises, and set the drill to students at the first four weeks of a semester (totally 12 hours). A number of small exercises are carefully created and ordered so that the students can start to solve them without any knowledge about programming at first (just only a little mathematical knowledge is required) and gradually understand the core of programming; evaluation of a statement, order of evaluation, assignment of a value, variables, self assignment of a variable, and the like that are common to all of procedural (and part of non-procedural) programming languages. We show the results of collected questionnaires about the drill as well as weekly tests.

2. Present ability

About 100 students took a programming classes and practices over two semesters (30 weeks in total, 45 hours for classes, 90 hours for practices) in the last year. In the practices, the students had been set several programming tasks including: computing Fibonacci sequence and factorial, calculating two fractions by structure, sorting, syntax analysis, and so on that are all well known programs.

To see their programming skill, several brief questions were set to them after the second semester was over. One of the questions is Fig.3 that is a simple rewriting from a mathematical equation into a representation in C code. Although this is a very easy question, the percentage of correct answers was 74%. Another question shown in Fig.4 asks to fill the

\[
x = \frac{a+b}{c-d(a+2)}
\]

\[
x=(a+b)/(c-d/(a+2))
\]

Fig. 3: Question: "rewrite the mathematical equation in C statement". (left) An equation in mathematics. (right) The answer : representation in C statement.

```
int i, sum=0;
for(____; ______)
    sum+=a[i];
printf("%d\n", sum);
```

Fig. 4 : Question "fill the blanks to sum up all elements of the array a[128]"
blanks to make the for statement work (the answer is "for \((i=0; i<128; i++)\)\). Understanding for statement is indispensable to make almost all program codes in any kind of procedural computer languages not limited to C, however, just only 41% of the students could answered correctly. We should note that the students were allowed to see any textbook on C when they answered the questions.

These easy questions may not support that the students have a little ability of programming, but undoubtedly reveal the absence of the fundamental knowledge about programming. All questions were such simple that they should be easily solved, however, the percentages of correct answers were between 20% and 78%, and sometimes fell to 5%. Some students said that they don't know that they can not use braces in C statement (only parenthesis are allowed) and that the index of elements in an array begins with 0 (not 1). Students should acquire these trivial but basic skills at the very early stage when they begin to learn programming, but only programming tasks in the practices could not achieve the goal.

3. Contents of drill in the proposed approach

The basic programming skill is common for all procedural languages; evaluation of a statement, assignment of a value into a variable, execution at each step is subject to control structures such as for, while, and if statements. These skill may be learned through drill in programming instead programming tasks.

However, such drill in programming have never appeared so far. The reason is the difficulty to extract the common basic concept of procedural languages by skillful programmers who have learned by themselves many techniques and facts that are implicitly supposed when they make a program. Another reason is that all textbooks assume that the reader tries to make a program in actual computer and run it. They believe that a computer is needed for learning programming, but training on programming can be done by only using a fraction of code on a paper, especially the fundamental knowledge.

In order to help students to acquire the basic knowledge, we have made nearly 230 small exercises that are carefully created and ordered so that the students can start to solve them without really any knowledge about programming at first (just only little high-school mathematical knowledge is required) and gradually understand the core of programming. Each exercise comprises of several (at most nearly 20) questions. Fig.5 shows some examples of the exercises. All 21 topics of the drill are listed in Fig.6 in order of appearance. Note that these topics take only one third part of ordinal textbook on C, and we focused only on the basic topics.

4. Practical training

To combine the drill and programming tasks, we divided the programming practice of a semester (15 weeks) into four periods. We have set the drill to the students in the first period (four weeks, a weekly practice is of three hours), and programming tasks in the rest three periods. The practice schedule of the periods is shown in Fig.8.
Fig. 7 shows the time table of a weekly practice of the drill period. Every week students had solved a fourth part of 230 exercises (about three minutes for one exercise in average), and at the end of the practice they answered some questions (like Fig.5) of the part they did in that day with the drill and a textbook as reference. Then at beginning of the practice in the next week, they answered questions of the part where they did in the last week, without any references this time.

Programming tasks in the periods include: hexadecimal-binary conversion, finding prime numbers, Euclidean algorithm, displaying filenames in a directory, etc.

5. Discussions

In this year, there are over 200 students for both class and practice, and most of them have no experience on programming.

After the drill period, the students tried to solve a short examination from all topics of the exercises. Each question is marked as correct or wrong (or partially wrong). The result is shown in Fig.9 where the percentages of each question is shown as a pair of two horizontal bars. The upper bar is of the first examination at just after the drill period, and the lower bar is of the second examination after two programming tasks passed (see Fig.8). Two examinations have the same contents and topics for comparison. As shown in Fig.9, the results of the examinations do not change so much. This means that the students became to be able to solve simple exercises (such as Fig.3 and Fig.4) by the drill, and the programming tasks do not improve the fundamental ability that appears just by the short examination. Note that by using a simple covariance structure analysis we confirm that the average of two examinations is the same but the variance increased (see Fig.10).

At the examinations, we collected questionnaires how much are you confident of understanding topics of the drill. The result of the questionnaires is shown in Fig.11. This illustrates that topics of double loop and infinite loop are hurdles for them to understand the control structure of programming, which have never suggested. The topics for loop have to be improved to make them understand well. The rate of “well” for array decreases drastically, which may be affected by the programming tasks they tackled in two periods. In the drill, exercises for array are so simple that they could solved easily, but in real programming they felt the difficulty of array (and pointer that is similar to but more difficult topic than array).

In another questionnaires, 95% of the students said that the drill was useful for solving programming tasks, and 62% that the drill period (four weeks) was relatively short, 82% that 230 exercises were too much to solve, 58% that the range of topics of the drill was small. Many students suggested the gap of level between the drill and the programming task; the most part complained “tasks are pretty harder than the drill", and a few said “the drill was easier than tasks”. Actually the level of the drill is lower than the tasks and far apart from real programming, but students demand lower level even if it may not be useful for programming. 5% students who answered that the drill was not useful for solving programming tasks suggested that point.

6. Conclusions

We have introduce a novel approach to teaching a computer programming language, especially C, to students in a programming practice, and reported examination results along with questionnaires of the drill. In our activity to inculcate very basics of programming to students, we created a huge number of small exercises that are carefully chosen and ordered so that the students can start to solve them without any programming knowledge and gradually under-
stand the core of programming that are common to all of procedural programming languages. According to the results of the examinations show that more training on loop is necessary than the current drill, we will restructure and reorganize contents and order of topics in the drill.

7. References


Curriculum Vitae

**Toru Tamaki:** Research associate, Information Science and Engineering, Graduate School of Science and Technology, Niigata University.

**Takeshi Hagiwara:** Research associate, Department of Information Engineering, Faculty of Engineering, Niigata University.

**Yoshinobu Maeda:** Research associate, Department of Biocybernetics, Faculty of Engineering, Niigata University.

**Yasuo Nakamura:** Research associate, Department of Biocybernetics, Faculty of Engineering, Niigata University.
Fig. 11: Questionnaire: how much understand the topics? (bar above) right after the drill period have done. (bar below) after two periods for programming tasks.

Fig. 10: A simple model for covariance structure analysis. Rectangles are observable variables, ellipses are hidden (latent) variables, circles are noise. The average of the ability comprises understanding and writing program did not change (from 0.0 to -0.2), while the std. increased to 1.07 from 1.