122nd ASEE Annual Conference & Exposition
June 14 - 17, 2015
Seattle, WA
Paper ID #12360

Effect of extended use of an executable flowchart for teaching C language

Prof. Cho Sehyeong, MyongJi University

Professor, Department of Computer Engineering, MyongJi University

Effect of extended use of an executable flowchart for teaching C language

Teaching computer programming to students is a daunting task, especially to those without any background or experience in computer programming. Even simple assignment statements or arithmetic operations can be difficult for them to understand. In our experience, roughly 25% of students fail the course and get frustrated that they are not fit for programming after all.

There are many reasons why programming can be so difficult for beginners. First, there are linguistic issues. The syntax of a programming language is very different from that of a natural language. Trivial grammatical errors can result in cryptic error messages that are hard to interpret. The students also encounter semantic difficulties. It is a challenge to get an accurate understanding of the operational semantics (i.e., effects) of the programming language constructs, which makes it difficult to predict the accurate result of a program code. This, in turn, makes it not so easy to write a program. Second, regardless of the difficulty of programming language at hand, the problem-solving process itself is inherently complicated.

In this paper, we hypothesize that the use of proper visual aid can improve students' learning speed and programming competence. One of the most popular visual aids for learning programming is a flowchart. In introductory programming courses, it is very common to explain the meaning of control structures, such as 'if-then-else' or 'do-while', by means of flowcharts. We propose to use an extended flowchart as an actual visual programming language, which is designed to enable smooth transition to commercial programming languages such as C, C++, or Java.

We developed a flowcharting tool that can be used for actual programming, as well as for executing, debugging, and visualizing. Thus, our specific aim was two-fold: first, help learn programming/problem solving and, second, facilitate the learning of a textual programming language – the C language. The actual hypothesis tested in the present study was X. The results of the experiment that was designed to test our expectation fully support our hypothesis. In what follows, we will briefly introduce the tool used and proceed with the discussion of the experiment and the results.

Related Work

There are many different approaches to facilitating the acquisition of programming language(s). For instance, in order to avoid the complexity of full-fledged programming languages, one can use simplified programming languages, such as Mini-Java[1]. In fact, mini-languages have been used for quite a long time[2]. However, this approach does not meet our requirements, since we

This work was supported by 2014 Research Fund of Myongji University.

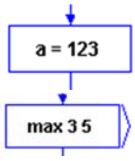
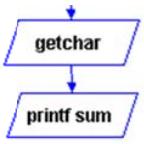
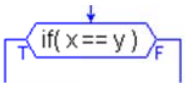
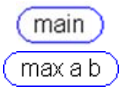
need to teach a full language after all. Iconic programming languages, such as Mindstorm NXT-G[3] or Alice[4], can be another possibility. Iconic programming is suitable for a gentle introduction to computer programming and it has been reported to help keep students interested[4]. Flowchart programming has been used elsewhere as an aid to help students understand the concept and to improve their problem-solving skills[5]. Raptor[6] is a flowchart programming tool used for teaching the concept and problem-solving skills, but transition from it to actual programming language such as C is time-consuming and requires extra effort.

A Brief Introduction to CFL

CFL, which stands for "C-like Flowchart Language," is an executable flowchart-based programming language and system. CFL is developed to help students learn programming, as well as to understand the mechanism of program execution. In particular, it is designed to help prospective C-language learners. A CFL program consists of nodes and directed arcs connecting the nodes. Currently, CFL version 2.0 has basic nodes and composite nodes. There are four types of basic nodes: processing, I/O, decision, and function nodes. Table 1 summarizes CFL simple node types. Composite nodes are used to group particular combination of basic nodes to give structures, such as "if-then-else", "for", "while", and functions. Structured programming is naturally enforced by composite nodes.

CFL is executable, and, therefore, has features related to execution. These features include: one accumulator register, one floating point accumulator, 12 integer variables that can be changed to float variables, two arrays, the input buffer, the output window, and two execution buttons – for single stepping and running/stopping. During the execution, students can watch the inner workings of the program: the control flow by a red dot and red-bordered node, changing values by flashing colors, and the function call stack by a stack of parameters. CFL is tightly integrated into a web-based instruction system for efficient assigning of exercises, submitting, and grading[7].

Table 1. CFL basic node types

type	processing node	I/O node	Decision	Function start
typical example(s)				
note	+, -, *, /, % function call, return	putchar, scanf printf	!=, ==, > ,<, >=, <= , &&, ||	only one main function

This work was supported by 2014 Research Fund of Myongji University.

For easier transition from CFL to C language, the syntax of CFL is intentionally aligned with C language syntax. These include arithmetic expressions, array notation, and input/output functions such as “scanf” or “getchar.” Control structures in C language such as “for,” “while,” and “if” are translated into intuitive graphical structures, as shown in Figure 1.

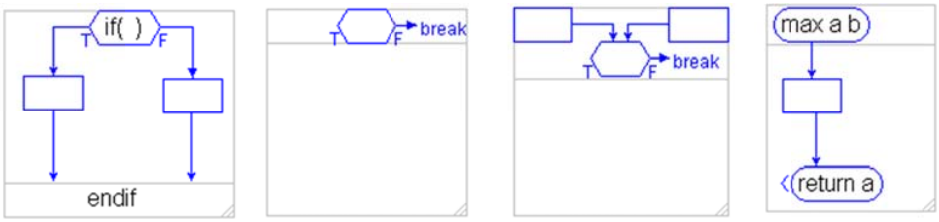


Figure 1. Composite Nodes: if, while, for, and function

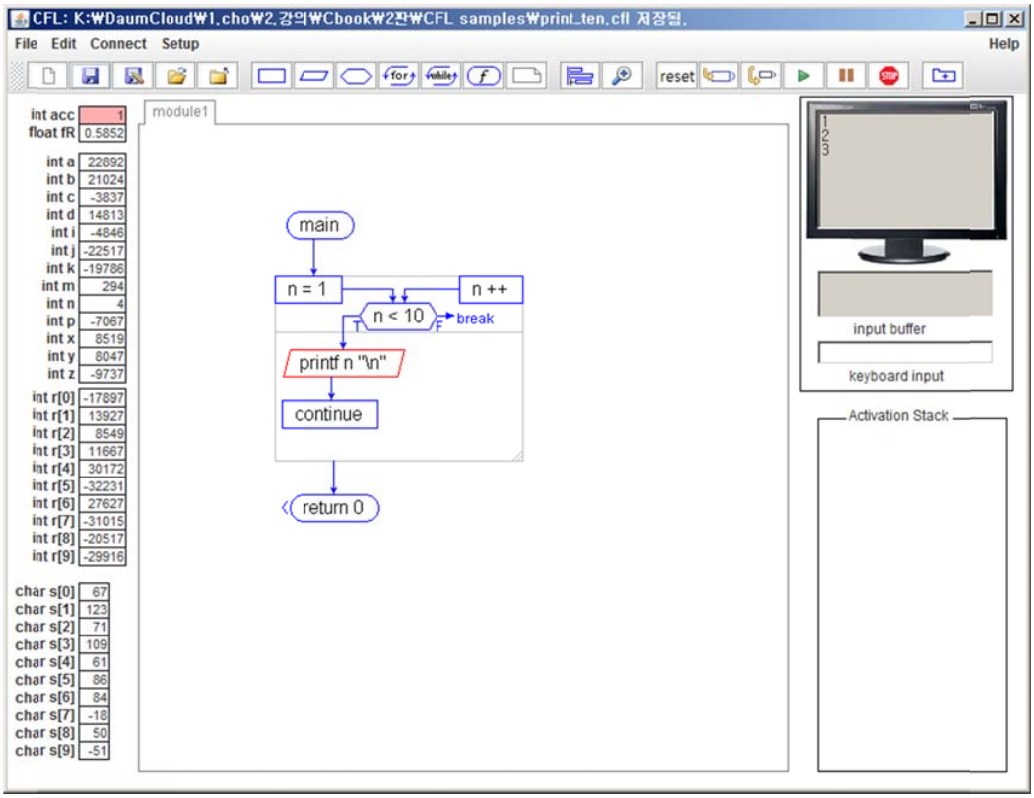


Figure 2. Snapshot of CFL editor/execution environment

The Experiment

It has elsewhere been observed that learning CFL control structures before learning C language

This work was supported by 2014 Research Fund of Myongji University.

control structures helps the C language learners learn faster and better[7]. The previous experiment was limited to a rather short time scale of a few hours. This time, we conducted an experiment to see if learning CFL has longer-term benefits in terms of weeks.

The preparation

We had two classes of the C programming courses with similar class sizes. Both classes started the semester with 35 students, i.e. the maximum number of students allowed in that semester. The students were taught by the same instructor on the same weekdays, but on different hours. The course was a 4 credit-hour course. The class met twice a week, each consisting of two consecutive class hours. One class - the control group - was taught by the regular C language syllabus. The other group – the treatment group – was first taught CFL programming for four weeks, before they started learning C language. Table 2 summarizes the semester schedule of the two classes. Although the course is intended for freshmen, there were quite a few students who were not freshmen. Since they were likely to have some experience in C programming, we excluded them from the analysis. This resulted in a total of 29 freshmen in the control group and 22 students in the test group.

Table 2. Summary of schedules

Week	Control group (C only)	Treatment group (CFL + C)
1	Intro to Computers, Prep for laboratory (incl. Linux and vim)	Intro to Computers, CFL basics, operations, I/O
2	Beginning C programming	CFL conditional, for loop
3	Integers and I/O	CFL arrays, functions and recursion
4	conditionals	CFL graphics and game project
5	while/for loops	Linux and vim, Integers and I/O,
6	Functions	conditionals, while/for loop
7	Arrays and applications	Functions, arrays
8	Handling strings, mid-term	arrays, mid-term
9	2-dim arrays	strings, 2-dim arrays
19	files	files
11	structures	structures
12	bit handling	bit handling
13	recursion	recursion
14	pointers and dynamic allocation	pointers and dynamic allocation
15	linked lists	linked lists
16	Final Exam	Final Exam

This work was supported by 2014 Research Fund of Myongji University.

To explore if learning CFL has any effects, either positive or negative, on learning C language programming, we tested the students with three identical tasks in the final exam, as well as observed their overall performance throughout the semester. The students were not informed about the ongoing experiment.

Task A

Problem description: Let X be a sequence of integers starting with 1, 2, 4, … and the differences between consecutive terms make an arithmetic progression of 1, 2, 3, …. Let Y be a sequence of integers that starts with 1, 2, 4, 8, … and each pair of consecutive numbers in the sequence has a difference defined by sequence X. Write a program to generate sequence Y less than 100.

Task A requires the understanding of how to write a loop. In the control group, 12 of 29 freshmen passed the test, compared to 15 of 22 in the treatment group. Thus, the performance in the treatment group was significantly higher than that of the control group (68.2% vs. 41.4%).

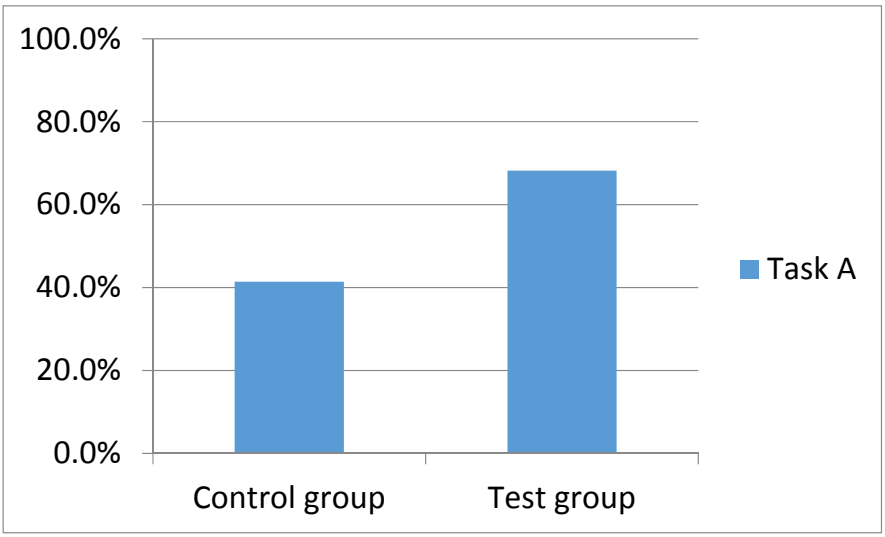


Figure 3.Performance of the two groups on task A

Task B

This work was supported by 2014 Research Fund of Myongji University.

Problem description:
Given an array of integers already in ascending order, write a program to insert an arbitrary new number into an existing set of numbers, such that the result is in an ascending order. However, you should follow the order given as follows: 1) determine what position the new number should go in; 2) shift all numbers greater than the new number; and 3) actually insert the new number.

The second task is a part of a sorting program that requires knowledge of and competence in handling 1-dimensional arrays. Both groups experienced 'bubble sort' during the semester. Sorting program is so simple that it is quite possible to memorize the code. In order to prevent "memorizing the solution", we provided specific constraints for sorting algorithm. In this test, the treatment group again outperformed the control group. Only27.6% of the freshmen in the control group solved the problem, while 45.5% in the treatment group solved it. Figure 4 compares the pass ratios for the second test.

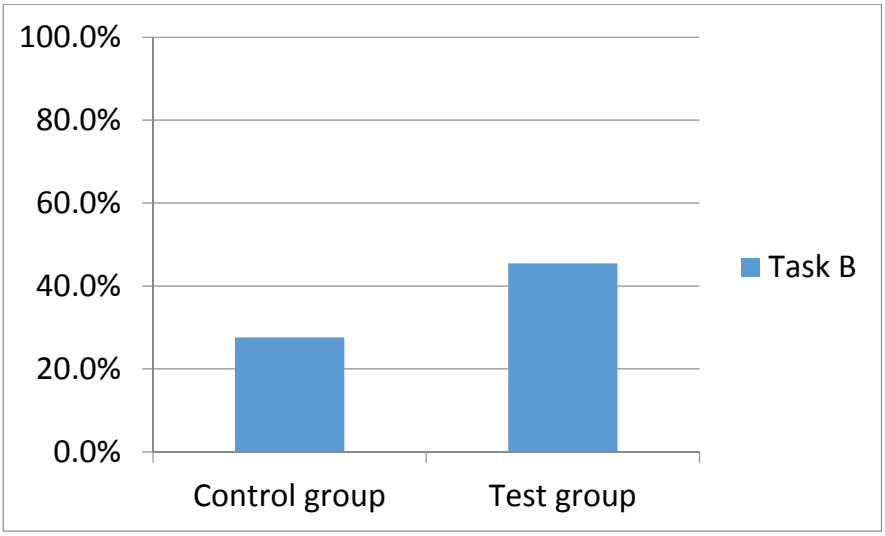


Figure 4. Performance of the two groups on task B

Task C

Problem description:
Given a representation of Omok (a.k.a., Gomoku) game in a 2-dimensional array, write a function to determine if a player has a winning configuration, i.e., 5 in a row.

Task C is to fill in some missing part of Omok game. For those who are not familiar with this game, omok is played by two players, black and white, taking turns to put a white or black

This work was supported by 2014 Research Fund of Myongji University.

stone until one of the player has 5 in a row, either straight or diagonally(see Figure 5).

This problem requires knowing how to deal with 2-dimensional arrays. On this task, the performance in the two groups was almost identical: 17 out of 29 students (58.6%) in the control group and 13 out of 22 students (59.1%) coped with the task.

This task differed from tasks A and B in that the students were given the same problem as homework assignment before, which might explain why the performance of the two groups was similar.

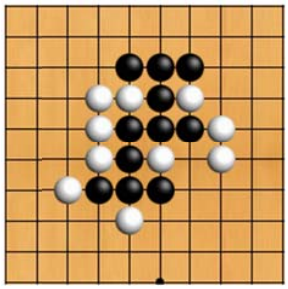


Figure 5. An Omok game

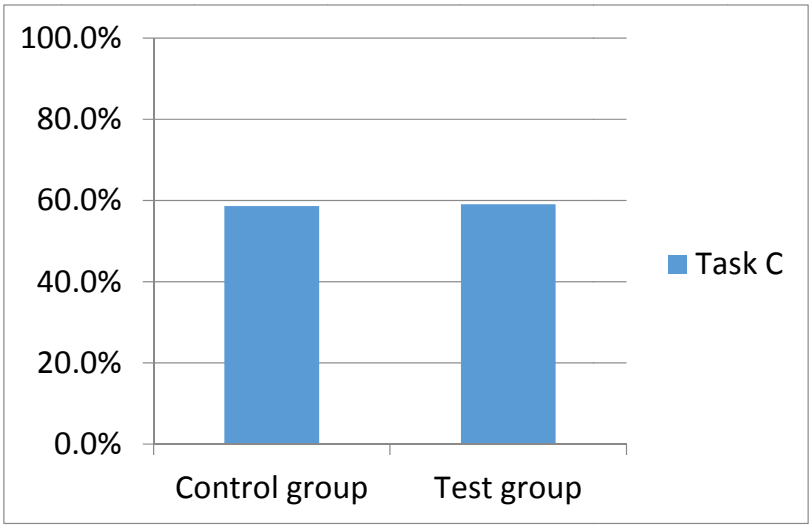


Figure 6. Performance of the two groups on task C

The overall performance

The treatment group outperformed the control group on two out of three problems in the exam. More importantly, in the end of the semester, far more students in the treatment group passed the course. Specifically, 90.9% of the treatment group (i.e., those who learned CFL before learning C language) passed the course. By contrast, only 72.4% of the control group passed the course. Grading is based on 15 problems in quizzes, midterm, and final exam, as well as homework

This work was supported by 2014 Research Fund of Myongji University.

assignments. The average scores of exams are 28.0 for control group and 40.9 for test group in a 100 scale. The pass ratio of the course in recent 3 years amounts to 74%. Therefore, 72.4% can be considered quite normal, while 90.9% can be considered exceptional.

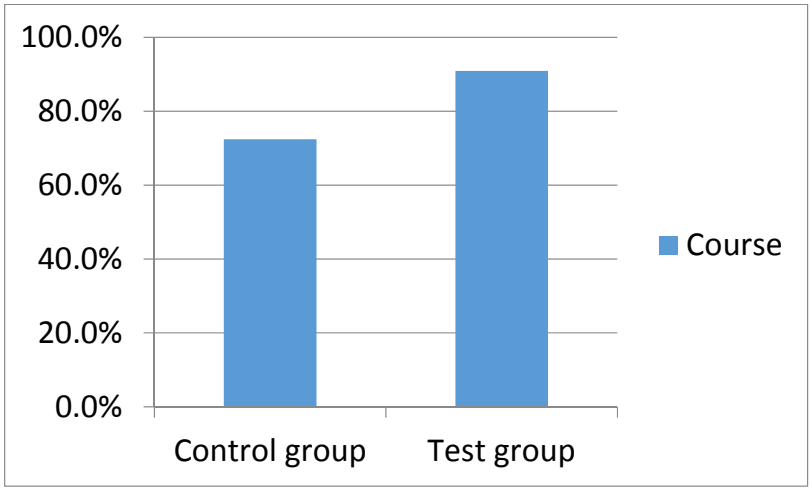


Figure 7. Course pass ratios of the two groups

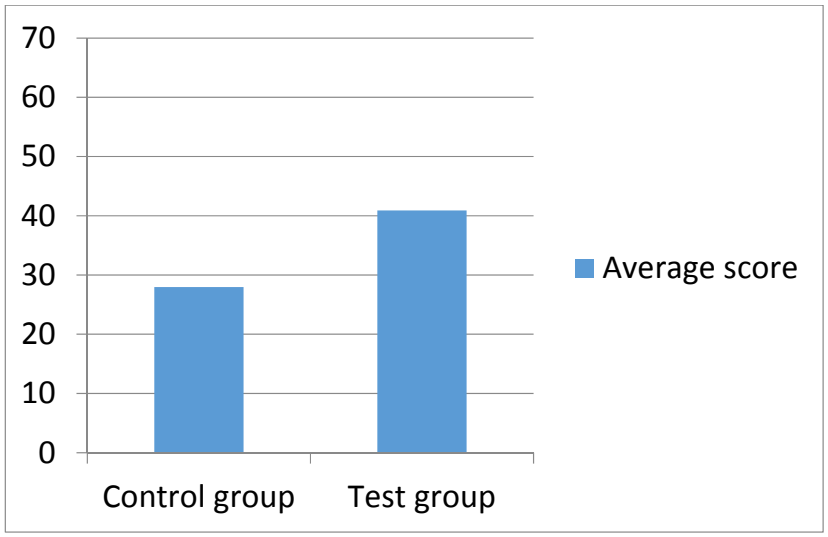


Figure 8. Average scores of the two groups

Conclusion and Future Work

To further previous research on short-term effect of using CFL for learning C language programming, the present study focused on corresponding long-term effects. Our results

This work was supported by 2014 Research Fund of Myongji University.

convincingly show that the students who are taught CFL before learning C language eventually outperform those who learned C programming in the first place. In view of the fact that both groups spent almost the same time learning, the results are even more encouraging.

We believe that the reason why we benefit from CFL is because CFL is in fact a C language in (graphical) disguise. We experienced that the students who learn CFL have more fun and do better in terms of "thinking" of the solution, because of the graphical nature of the language and the execution environment. Furthermore, the competence acquired with CFL does not seem to be diminished later by the complexity of the C language syntax, for once one understands how to do something in CFL, it is easy to translate that into C language syntax. However, we cannot completely rule out the possibility that the seemingly encouraging result has been obtained purely by chance. Therefore, another experiment is planned for the coming semester.

References

1. Roberts, E. "An Overview of MiniJava", *ACM SIGCSE Bulletin* 33 (1), 2001, pp. 1-5.
2. Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A., and Miller, P. "Mini-languages: A Way to Learn Programming Principles", *Education and Information Technologies 2* (1), 1997, pp. 65-83.
3. Swan, D. "Programming Solutions for the LEGO Mindstorms NXT," *Robot magazine*, 2010, p. 8.
4. Sattar A., Lorenzen T. "Teach Alice programming to non-majors", *ACM SIGCSE Bulletin 41*(2), pp. 118-121.
5. Crews, T. "Using a Flowchart Simulator in a Introductory Programming Course", http://www.citidel.org/bitstream/10117/119/2/visual.pdf Last accessed Dec.30, 2013.
6. Martin C., Carlisle et al. "RAPTOR: introducing programming to non-majors with flowcharts", *Journal of Computing Sciences in Colleges* 19(4), 2004, pp. 52-60.
7. Sehyeong Cho, Ryu, Y. S., and Kim, S. "Learning C language programming with executable flowchart language", *Proceedings of ASEE annual conference*, 2014, paper ID#8872.

This work was supported by 2014 Research Fund of Myongji University.