Comprehension and Composition of Flowcharts

Unaizah Obaidellah
Faculty of Computer Science & IT
University of Malaya, Malaysia
unaizah@um.edu.my

Abstract

The common problem of the students' failure in programming courses is due to the lack of problem solving skills. This is regarded as the lack of ability to understand the problems, divide them in sub-portions, and integrate them as a complete solution. The main purpose of this paper is to present preliminary findings of students' comprehension and composition abilities focusing on problem solving activities related to interpreting code into flowcharts and flowcharts into code. The group of students tested were undertaking their second programming course. Initial findings from this study showed that the students possessed relatively weak problem solving skills even if they scored well in their fundamental programming course.

Keywords: Program comprehension, flowchart, data structure, problem solving

1. Introduction

Computer programming is a fundamental course of the computer science curriculum. A trend of high drop out and failure rate in introductory computer programming courses in universities is a universal problem (Lahtinen, Ala-Mutka, & Järvinen, 2005; Mayer, 1981; Mhashi & Alakeel, 2013; Milne & Rowe, 2002; Mow, 2008; Tan, Ting, & Ling, 2009). The majority of these research works performed a survey study that focuses on students' perceptions of difficulty in understanding and implementing both low-level (i.e., syntax, variable) and high-level concepts (i.e., OOP, debugging). A general conclusion produced from these studies showed that learning programming is a difficult task to master by novice programmers. This difficulty is attributed by various aspects including teaching methods, study methods, student's ability, motivation and nature of programming (Gomes & Mendes, 2007; Renumol, 2009; Sweller, 1994).

The study methods adopted by many students such as incorrect study methodologies and less intensive programming exercises contribute to this difficulty. Furthermore, the student's abilities and attitudes towards programming such as the lack of problem solving skills required to interpret a particular problem statement, and limitation in mathematical and logical knowledge necessary to transform textual problem into successful solution code, are among the factors that reduce the students' motivation to study programming. In addition, a pessimistic mindset associated with programming that spreads among the students sets an established reputation of programming courses being a difficult subject.

Success in programming is further attributed by factors such as prior computing experience, strong mathematical skills, learning style, gender and culture (Butler & Morgan, 2007; Lister et al., 2004; Mhashi & Alakeel, 2013; Piteira & Costa, 2013; Renumol, 2009). Although, these factors, taken independently or dependently, arguably influence a student's programming ability, many previous research outcomes indicated that one of the major reasons students have difficulties programming is due to their lack of skills to solve problems. This is demarcated by insufficient set of abilities necessary for solving problems such as abstraction, generalization, transfer and critical thinking (Gomes & Mendes, 2007). These abilities are associated with limited understanding of how a program is executed (Milne & Rowe, 2002) which includes the ability to understand a problem description, separate it into sub-portions followed by an implementation before assembling these pieces into a complete solution. Lister et al. (2004) asked individual undergraduate students to study short computer programs and answer multiple choice questions in an experiment that examines the processes involved in producing the solutions. Their study proposed that the inadequate knowledge and skills relate to the students ability to read code rather than writing which concluded that students often fail to analyse a short piece of code. However, this result is limited to reading code and choosing from a set of pre-defined answers. This approach may limit the results as students may use different mechanism for tasks such as writing code. Thus, considering the proposed future work by Lister et al (2004), the present study aims to evaluate students' ability reading and writing code with respect to flowcharts (see below for flowcharts consideration). Both of these tasks can be classified as program comprehension (i.e., understanding what a program does such as aims and processes) and program generation (i.e., writing program code for a given programming problem). Robins, Rountree, & Rountree (2003) proposed that these tasks operate on different models. By this, novices tended to create program plans, while experts retrieve them, requiring explicit attention during planning and problem solving among novices. Thus, Robins et al (2003) proposed some level of dependent relationship between program comprehension and generation although they noted that these tasks may not be well correlated.

Given this unclear relationship, the proposed research idea is reported as part of an ongoing study in programming education. This work-in-progress report is presented as part of the main goal to better understand novice learners' lack of skills at solving problems in the context of advanced programming course (i.e.: Data Structure) and to provide recommendations (i.e.: learning model/framework of cognitive model) for novices to aid learning. As discussed, the preliminary focus of this study is an extension of Lister et al. (2004), where students are evaluated on their competency reading and translating flowcharts into program code and vice versa. Based on our teaching experience, this evaluation is considered important because often students report sufficient understanding of programming concepts but fails to demonstrate writing its equivalent programming code. This raise questions whether this issue is influenced by weak fundamental or advanced programming concepts, cultural impact or other unclear reasons. Thus, better understanding on this lack of skills would contribute to the development of improved problem solving strategies.

Our motivation for the preliminary study described in this report was to assess the students' composition and comprehension skills among those who passed the prior fundamental programming course. Specifically, investigation focuses on whether these skills would benefit novice learners to facilitate the solution process of a programming problem on a data structure topic (i.e.: linked list). In this context, comprehension refers to the students' understanding of the given flowchart or program code and recognizing its relevant purpose. Composition refers to the students' ability writing code with respect to the given flowchart or producing flowcharts from a specified set of functional code. Thus, this preliminary study will inform about students competency level at solving advanced programming problems. The proposed evaluation is different than those reported in existing literature as the student participants are assumed equipped with relatively strong fundamental programming concepts (i.e.: variables, control structure, arrays, methods, OOP) commonly taught in the first programming courses as demonstrated by good grades acquired by majority of the students in their final examination results of the fundamental programming course.

2. Flowcharting and code comprehension

In programming, flowcharts are commonly introduced to early learners to facilitate the understanding of a problem and segments of the solution processes. Flowcharts are useful to aid the creation of solutions to problems or facilitate the construction algorithms. In addition, flowcharts would serve as a foundation in drafting solutions for a programming problem. However, this tool is less practiced in many cases especially by novice learners. Students jump straight into writing the solution code. Our observation showed that students often attempt to solve a problem almost immediately after a short contact with the problem description, even before careful analysis and understanding of the question requirement. This problem conforms with the study reported by Almeida (2004).

Another empirical study by Shneiderman & Mayer (1979) investigated the effects of flowcharts on students' performance in comprehension and debugging tasks between two different universities. A closer inspection showed that better performance was reported for the university which training emphasized the use of flowcharts although generally, no difference in performance was found. However, Shneiderman & Mayer (1979) concluded that flowcharts may serve either as an advantage where solution is an aid that translates the process from syntax to semantics or a disadvantage where flowcharts were taken as an alternative representation which interferes with the students existing semantic mental structure. As such, what types of supplementary representation help programming learners to build their internal semantics was raised as a question worth further investigation. Furthermore, various levels of abstraction can occur in a flowchart construct due to the richness of consideration for a programming solution. It will be interesting to investigate whether some level of generality occurs between reading code and producing the corresponding flowcharts and vice versa.

2.1 Hypothesis

It is assumed that if a student can consistently demonstrate an understanding of flowcharts by naming an appropriate function name for the given problem followed by writing the equivalent program code, then it is estimated that a student has adequate problem solving skills. Similarly for understanding code followed by drawing the flowchart. However, if a student can demonstrate comprehension at naming the method or flowchart, but not able to write the program code, it is reasonable to conclude that such students lack the knowledge and skills for problem solving. Similar principle is applied if a student is not able to either, name the method, draw the corresponding flowchart or write the appropriate program code.

3. Data collection

Data for this study was collected under exam conditions. The linked list data structure topic was tested. The students were taught this particular topic and completed a tutorial and a lab session prior to the test. Generally, the students learned the use of standard notational description of the basic shapes of flowcharts in their fundamental programming course. During teaching, the students were shown how these flowcharts were used in linked list.

A total of 90 students contributed data to this part of the study. All students answered two questions – the first question on flowchart and the second on programming code presented in Java where necessary. The difficulty level for each question was considered equal as the two are developed based on slight variations of one another. This means the answer for question 1 is somewhat identical to the given flowchart from question 2. Similar principle applies to the programming code posed in question 1.

The open-ended questions required students to study a flowchart or a short source code of a method (i.e., getElement and contains methods) and provide answer in two parts, a) name the operation of the method or flowchart, b) draw the corresponding flowchart or write the code also in Java. Score out of 10 marks was calculated - one each for naming the code/flowchart and four marks for each correctly interpreted question. As for part (a) - naming the code, a full mark is given if the answer closely represents the processes defined by the code/flowchart showing that a correct answer is provided. Thus, the given method name answered need not exactly match the marking scheme. Ambiguities are resolved by validating with the corresponding answers provided in part (b) where a full mark is given for part (a) if the code/drawing in part (b) are correct. The four marks for part (b) were divided by percentage of correctness with 25% for 1 mark and increase by an additional mark for every further 25%. Zero mark is given if an answer is unrelated or incorrect.

All students who undertook this test had taken a fundamental programming course studied in Java in their first semester. A large majority of them scored grade A in their final examination. They were undertaking the Data structure and algorithm course as advanced programming, a continuation programming course in their second semester of their first year at the time of the test.

4. Performance data analysis and discussion

The scores of the students are shown in Table 1 and 2. Figure 1 and 2 show the corresponding graph representation.

Q1	**Flowchart Composition**				
Code Comprehension	**0**	**1**	**2**	**3**	**4**
	Very weak	Weak	Average	Satisfactory	Excellent
Poor (0)	1	3	6	10	18
Excellent (1)	0	5	4	7	36

Comprehension. Poor = 0 mark ; Excellent = 1 mark

Composition. Very weak = 0; Weak = 1; Average = 2; Satisfactory = 3; Excellent = 4 mark

Table 1- Relationship between comprehension of code and composition of flowchart for Question 1

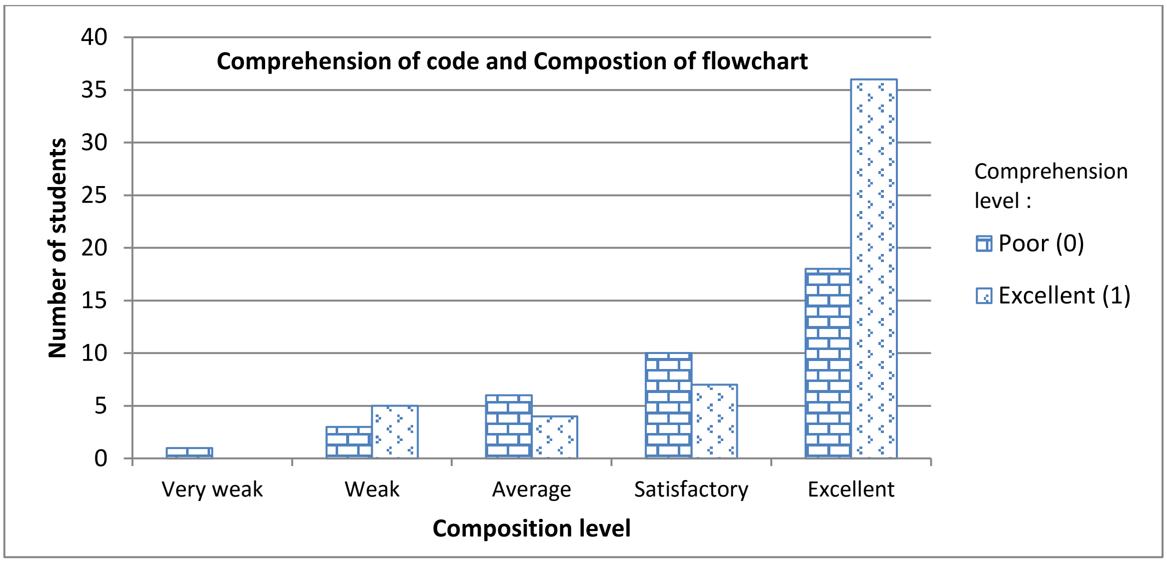


Figure 1- Students' success rate at understanding the code and drawing the flowchart for Question 1. The comprehension level is represented by poor(0) and excellent(1) performance.

In question 1, the students' performance analysis in terms of their ability to comprehend or understand the code (remarked by excellent comprehension in Table 1 and Figure 1) showed that majority of the students who named the method correctly could draw excellent flowchart equivalent to the given code. Similarly, those who did not manage to name the method correctly (remarked by poor comprehension) remained successful at drawing the flowchart, although the number of students for this dropped by half. However, a paired t-test between the code comprehension and flowchart composition did not show a statistically significant results. Thus this may not support our hypothesis fully that those who are competent at understanding the program code in terms of its functions and name the appropriate method name are able to compose the flowchart without difficulties.

Q2	**Code Composition**				
Flowchart Comprehension	**0**	**1**	**2**	**3**	**4**
	Very weak	Weak	Average	Satisfactory	Excellent
Poor (0)	6	25	9	3	1
Excellent (1)	0	3	14	25	4

Comprehension. Poor = 0 mark ; Excellent = 1 mark

Composition. Very weak = 0; Weak = 1; Average = 2; Satisfactory = 3; Excellent = 4 mark

Table 2 - Relationship between comprehension of flowchart and composition of code for Question 2

Analysis of question 2 showed that 25 students who could not interpret the flowchart (i.e.: not able to name the process workflow) adequately were weak coders as they only managed to write approximately 20% of the correct source code, thus, scored a 1 mark. Surprisingly, majority of the students who managed to name the process workflow of the flowchart were only able to write satisfactory codes. This means a large portion of the written source code were correct (i.e.: 75% correct), but contained inaccurate lines with respect to the given algorithm. Only four students showed excellent codes relative to the flowchart of question 2. A paired t-test comparison between flowchart comprehension and code composition showed a significant results, p<.05. This shows that there is a strong relationship between flowchart comprehension and code composition indicating that good proficiency reading flowcharts would at least improve a student’s competency at implementing program codes.

Based on this preliminary investigation, between these results, it is reasonable to conclude that understanding programming code followed by drawing its equivalent flowchart seems easier compared to the opposite task. In other words, the statistical distribution of the code writing performance showed the mean towards excellent composition for the excellent comprehension group and with the mean towards very weak composition for the poor comprehension group, respectively. The students can be considered better problem solvers when translating code into flowcharts. However, their problem solving skills require more comprehensive knowledge, practice and skills necessary for understanding a flowchart before converting them into the respective code. A paired t-test between both questions compared between each composition level was non- significant.

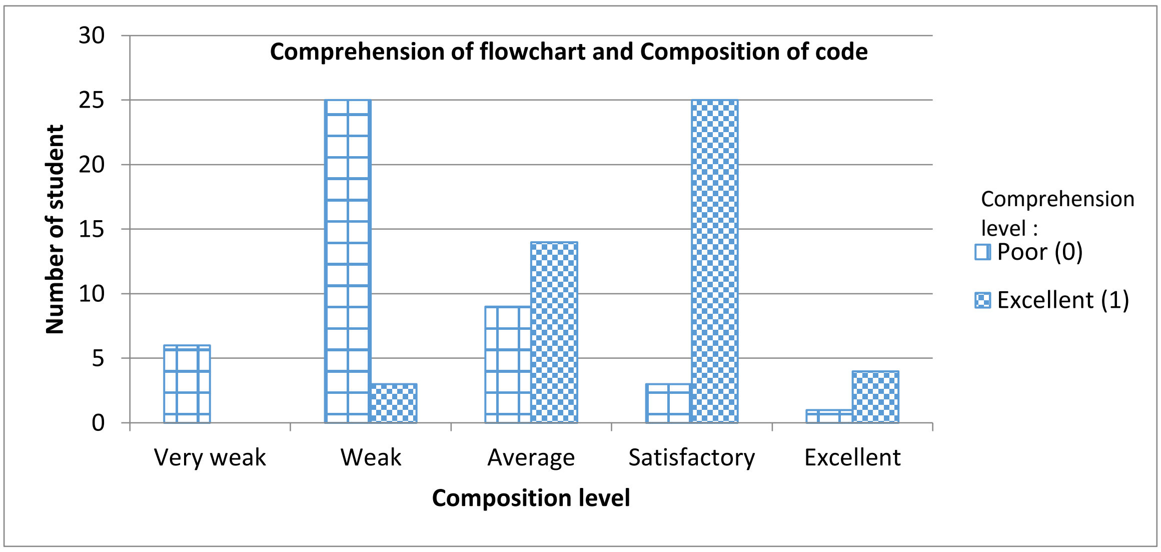


Figure 2 - Students’ success rate at understanding the flowchart and writing the code for Question 2. The comprehension level is represented by poor(0) and excellent(1) performance.

5. Conclusion

This preliminary study investigates first year Computer Science students' level of comprehension and composition by analysing tasks related to interpreting code into flowcharts and flowcharts into code. These students were tested on linked list, a data structure topic they studied in their second programming course (CS2). The early findings showed that they are still weak in coding and solving programming questions although a large number of these students did well in their first programming course. This finding is consistent with that reported by Lister et al. (2004) who proposed that student who are weak at reading code are potentially also weak in problem solving. This translates to an interpretation that if students are unable to read code well, it is unlikely that they will do well in writing novel codes. The reasons for this lack of skills may be attributed by vast possibilities such as lack of comprehensive conceptual understanding of the data structure topic, students do not master the flowcharting knowledge and weak at coding or uses inappropriate syntax while solving a programming problem. The initial result of this study found that these students perform better in translating source code into flowcharts more than the opposite task. This suggests a few possibilities including 1) the flowchart notation may be harder to understand than the Java syntax, at least for the problem studied, 2) the nature of Java as an imperative language enables a short Java program as that tested in the present study to be automatically divided into short segments which can be individually examined. However, a flowchart taken as a large undifferentiated whole which has to be understood all at a time may be a contributing factor for the differences in both tasks. 3) the Java language in the form of syntax uses conventional lexemes for control structures and other operations allows immediate understanding of the corresponding semantics. In contrast, flowcharts do not have lexical representation for familiar control structures apart from the standard notational shapes (i.e., rounded rectangle for start/end program, rectangle for processing, diamond for decision, parallelogram for program input/output) that is less informative than the kinds of control/selection structures represented in codes (i.e., for loop).The lexical representation have to be inferred from the connectivity between the shapes which can cause further difficulties as the parts of a flowchart can be spatially rearranged as long as the connectivity maintains. For this reason, a flowchart reader cannot necessarily rely on all types of loops structured in the same pattern. Thus, generally, this interpretation suggests that the flowcharts are a lower-level notation than the Java codes and the use of flowcharts for the type of task used in this study may be a causal role in the task being harder. Given this limitation, the next logical research step is to improve the notation used in this study. Instead of using flowcharts, a higher-level notation should be considered. The results of the present study could be better analysed using a more standardized coding scheme such as Good and Brna (2004), that was developed by analysing levels of abstraction in program summaries that allow participants to express their understanding in their own words. The coding scheme developed based on Pennington's (Pennington, 1987) showed how descriptions of programs are classified. Another limitation of this study is that only one question for each type of translation was tested. Thus, the present results only apply to the questions chosen. Hence, limits the generalization of the results. Less emphasize on flowcharting given during lecture and practical exercises could have contributed to the student's ability in comprehending the flowcharts as expected.

This inconclusive result stipulates further extensive study. It is proposed that the following is considered in the upcoming study:

- Develop measures to specify what and how problem solving approaches can assist beginner learner to develop their semantic knowledge.
- Evaluate the effectiveness of visualization in programming education by engaging learners in active learning activity.
- Determine whether students who perform in comprehension of code and flowcharts would do well in other programming activities such as commenting code, completing incomplete portion of code, memorization, debugging, modification, restructuring a random process flow of algorithm.
- Specify what learning activities facilitate novice learners to develop their problem solving skills.

- Identify what cognitive processes are involved in problem solving and coding by testing and comparing existing learning models.
- Analyse the levels of abstraction in programming comprehension according to the coding schemes proposed by Good and Brna (2004).

6. Acknowledgement

This research was in part supported financially by the University of Malaya Research Grant (RP030C-14AET) and Fundamental Research Grant Scheme (FP062-2014A). The author is grateful for all colleagues from Faculty of Computer Science, University of Malaya, who provided insight and expertise that greatly assisted the research.

7. References

Almeida, A. C. (2004). *Cognição como Resolução de Problemas: Novos horizontes para a investigação e intervenção em Psicologia e Educação.* Faculdade de Psicologia e Ciências da Educação da Universidade de Coimbra.

Butler, M., & Morgan, M. (2007). Learning challenges faced by novice programming students studying high level and low feedback concepts. *Proceedings of the 24th Ascilite Conference*, 99–107. Retrieved from http://ascilite2012.org/conferences/singapore07/procs/butler.pdf

Gomes, A., & Mendes, A. (2007). Learning to program-difficulties and solutions. *International Conference on Engineering Education–ICEE., 2007.* Retrieved from http://ineer.org/Events/ICEE2007/papers/411.pdf

Gomes, A., & Mendes, A. (2007). Problem solving in programming. *The Proceedings of PPIG as a Work in Progress Report*, 216–228. Retrieved from http://www.ppig.org/papers/19th-Gomes.pdf

Good, J., & Brna, P. (2004). Program comprehension and authentic measurement: A scheme for analysing descriptions of programs. *International Journal of Human Computer Studies*, *61*(2 SPEC. ISS.), 169–185. doi:10.1016/j.ijhcs.2003.12.010

Lahtinen, E., Ala-Mutka, K., & Järvinen, H.-M. H. (2005). A study of the difficulties of novice programmers. *ACM SIGCSE Bulletin*, *37*(3), 14–18. doi:10.1145/1151954.1067453

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., … Seppälä, O. (2004). A Multi-National Study of Reading and Tracing Skills in Novice Programmers. *ACM SIGCSE Bulletin*, *36*(4), 119–150.

Mayer, R. E. (1981). The Psychology of How Novices Learn Computer Programming. *ACM Computing Surveys*, *13*(1), 121–141. doi:10.1145/356835.356841

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B. D., … Wilusz, T. (2001). A multinational, multi-institutional study of assessment of programming skills of first-year CS students. *SIGCSE Bulletin*, 125–140. doi:10.1145/572139.572181

Mhashi, M. M., & Alakeel, A. L. I. M. (2013). Difficulties Facing Students in Learning Computer Programming Skills at Tabuk University. *Recent Advances in Modern Educational Technologies*, 15–24.

Milne, I., & Rowe, G. (2002). Difficulties in learning and teaching programming—views of students and tutors. *Education and Information Technologies*, *7*(1), 55–66. Retrieved from http://link.springer.com/article/10.1023/A:1015362608943

Mow, I. (2008). Issues and difficulties in teaching novice computer programming. *Innovative Techniques in Instruction Technology, E-Learning, E-Assessment, and Education.*, (Springer Netherlands), 199–204. Retrieved from http://link.springer.com/chapter/10.1007/978-1-4020-8739-4_36

Piteira, M., & Costa, C. (2013). Learning computer programming. In *Proceedings of the 2013 International Conference on Information Systems and Design of Communication - ISDOC '13* (p. 75). New York, New York, USA: ACM Press. doi:10.1145/2503859.2503871

Renumol, V. (2009). Classification of cognitive difficulties of students to learn computer programming. *Indian Institute of Technology*. Retrieved from http://dos.iitm.ac.in/publications/LabPapers/techRep2009-01.pdf

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2), 137–172. Retrieved from http://www.tandfonline.com/doi/abs/10.1076/csed.13.2.137.14200

Sheard, J., Simon, Hamilton, M., & Lönnberg, J. (2009). Analysis of research into the teaching and learning of programming. *Proceedings of the Fifth International Workshop on Computing Education Research Workshop.*, (ACM), 93–104. doi:10.1145/1584322.1584334

Shneiderman, B., & Mayer, R. (1979). Syntactic/semantic interactions in programmer behavior: A model and experimental results. *International Journal of Computer & Information Sciences*, *8*(3), 219–238. doi:10.1007/BF00977789

Sweller, J. (1994). Cognitive load theory, learning difficulty, and instructional design. *Learning and Instruction*, *4*, 295–312. Retrieved from http://www.sciencedirect.com/science/article/pii/0959475294900035

Tan, P.-H., Ting, C., & Ling, S. (2009). Learning difficulties in programming courses: Undergraduates' perspective and perception. *2009 International Conference on Computer Technology and Development*, *1*(IEEE), 42–46. doi:10.1109/ICCTD.2009.188

8. Appendix

Questions were given to students in printed copies where they were asked to answer in empty spaces allocated in the question paper. Here, the spacing is altered to fit the conference format. A short description and notation of the basic shapes (i.e., rounded rectangle for start/end of a program, rectangle for processing, diamond for decision, parallelogram for program input/output) used to represent standard flowcharts was given prior to the questions.

Question 1

Given are codes for OperationX(int index). Answer the following questions:

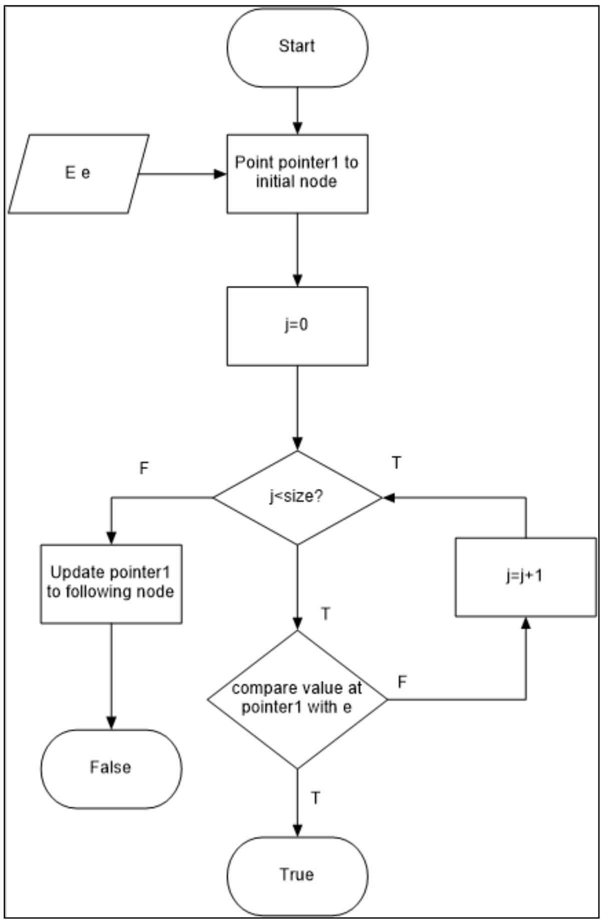
```
public E OperationX(int index) {
      Node<E> temp = head;
      for(int i=0; i<index; i++) {
         temp = temp.next;
      }
      return temp.element;
    }
```

a) What is the name of the Operation X?

b) Draw the flowchart for OperationX(int index).

Question 2

Study the following flowchart.



Start
E e
Point pointer1 to initial node
j=0
F
T
j<size?
Update pointer1 to following node
j=j+1
T
compare value at pointer1 with e
F
False
T
True

a) What is the name of the Operation Z?

b) Using the method name you give in 2(a), convert the flowchart given above into code. Write the method signature with an appropriate method name.