

**School of Education  
STEM Education Research Group**

**Teaching and Learning Introductory Computer Programming at  
the Royal University of Bhutan: Factors Affecting Student  
Performance**

**Mani Pelmo**

**This thesis is presented for the degree of  
Doctor of Philosophy  
of  
Curtin University**

**December 2019**



## **Declaration**

To the best of my knowledge and belief, this thesis contains no material previously published by any other person except where due acknowledgement has been made.

This thesis contains no material that has been accepted for the award of any other degree or diploma in any university.

Signature:

Date: 12 December 2019



## Abstract

Worldwide, learning computer programming is considered difficult for novice students when they are first exposed to it at university. Due to this, students may become demotivated, lose interest and slowly withdraw from programming units. Thus, there tends to be high attrition rates in early tertiary-level programming units; this is a well-known problem in computing education. Previous studies have investigated the cause of the difficulties in learning to program and have identified a number of curricular approaches to assist students to overcome these. However, the problem still exists, suggesting the need for further research.

This study investigates the factors that may affect the performance of students studying the unit ‘Introduction to Computer Programming’, henceforth referred to as CS1 in the country of Bhutan. At the Royal University of Bhutan (RUB), CS1 is the first and compulsory module studied by students enrolled in science and engineering programs. These students find programming difficult, which has led to high failure rates in CS1 at RUB. This high failure rate and students’ inability to comprehend the programming concepts at RUB is a cause of concern for both for the researcher and the institution. Thus, the researcher has set out to investigate the factors that may affect student performance in CS1.

The conceptual framework for this study was adapted from Biggs 3P model of learning to describe the factors that may affect students’ performance in CS1. The first P (input factors) describes the factors that students bring before the commencement of CS1 and the institutional factors that are

already in place. The input factors discussed in this thesis are prior computing experience, Year 12 performance in mathematics, physics and chemistry, and institutional factors are teaching methods and practices, programming paradigm, programming environment and language used in CS1. The second P (learning process factors) describes the learning environment in which CS1 takes place. The learning process factors discussed in this thesis are students' learning approaches, students' ability in programming skills, students' learning methods and practices. The last P (student performance) is the final outcome after learning has taken place. Student performance in this study was measured by student scores in the final semester examination, overall semester performance and a researcher-designed programming skills test. The programming skills test was devised to measure students' ability across five programming skills, identified in this study as algorithm design, translating, tracing, explaining and writing. The structure of the observed learning outcomes was adapted to determine in-depth information about the students' level of understanding in the test. The relationships among the programming skill variables and between student performance and programming skill variables were examined in this study. Multiple regression analysis was conducted to investigate the relative contribution of each of these programming skill variables to student performance. The results of multiple regression analysis and the path analysis reported that 48 percent of the variance was explained by the programming skill variables, in which writing contributed the most and algorithm design the least to student performance.

This study employed a convergent parallel mixed methods design and a variety of data collection methods including a test, survey questionnaire, and individual and group interviews designed to address the research questions. The qualitative data were used to confirm/disconfirm the quantitative data and to provide deeper meaning. Participants for this study were students and lecturers learning and teaching CS1 in July–November 2016 at three colleges of RUB, which are geographically separated by rugged terrain. A total of 292 students took the test and 277 participated in the survey. All eight lecturers participated in a survey and an interview.

This study is significant, as it is the only study to investigate the variables discussed in this thesis in the country of Bhutan. Thus, it will have implications for the teaching and learning of CS1. The outcome of the study will provide CS1 lecturers at RUB with greater insight into their teaching practices and present them with data to inform curricula decisions as they work to improve student performance.

The results from this study will also provide a foundation for further research in terms of the new variables introduced that are important in teaching CS1 as well as draws further attention to the current variables for existing computing tertiary educators.





## Acknowledgements

First, I would like to thank the Schlumberger Foundation, Faculty for the Future for sponsoring my studies for four years, and Curtin University for the final five months. I would also like to thank the Royal University of Bhutan (RUB) for its support in the scholarship application process and for granting study leave. Thank you also to Curtin University for accepting me and providing resources. Without its support, my PhD would have remained a dream. Thank you to the Schlumberger Foundation, RUB and Curtin University for realising my dream to study for a PhD.

Finally, I would like to acknowledge and thank the following people who have been part of my journey. Professor David Treagust, of Curtin University, provided valuable support and help during the application process to Curtin University and Schlumberger Foundation. Without his commitment, I would not be where I am today. Thank you also for being my co-supervisor and rendering your assistance in times of need.

Dr Tony Rickards, my ex-supervisor, thank you for your unwavering support, guidance, encouragement and motivation during my initial year. I have been lucky to work with such a caring supervisor who cared not only about my work, but also my wellbeing and family.

Dr Martin Cooper, my primary supervisor at Curtin University, thank you for your patience and for taking responsibility for my work. Thank you for being positive and guiding me throughout this research. I have always appreciated your advice and the support you have given. Thank you, it was a privilege working with you.

Thank you to Dr Lisa Lines of Capstone Editing for proofreading and editing my thesis according to the guidelines laid out in the university-endorsed national ‘Guidelines for Editing Research Theses’.

Rinzin, my loving husband, thank you for being next to me throughout my journey, providing unconditional support and encouragement to keep me going. Also, I sincerely thank you for looking after our two beautiful daughters, Yangchen and Seday. Yangchen and Seday, thank you for understanding when I could not attend your functions and events at school or take you to parks and swimming pools. Remember, you are the reason I wake up every day.

My loving parents back in Bhutan, thank you for having faith in me and showing that nothing is impossible in life if only you try.

Last but not the least, I sincerely thank all lecturers at RUB who assisted in my research. My sincere thanks also goes to all friends and colleagues who have been part of my journey.

# Contents

<b>Declaration</b> .....	<b>iii</b>
<b>Abstract</b> .....	<b>v</b>
<b>Acknowledgements</b> .....	<b>ix</b>
<b>Contents</b> .....	<b>xi</b>
<b>List of Figures</b> .....	<b>xv</b>
<b>List of Tables</b> .....	<b>xvii</b>
<b>List of Abbreviations</b> .....	<b>xix</b>
<b>Glossary of Terms</b> .....	<b>xxi</b>
<b>Chapter 1: Introduction</b> .....	<b>1</b>
1.1 Introduction .....	1
1.2 Thesis Origin .....	1
1.3 Background .....	2
1.4 Research Objectives .....	9
1.5 Significance of this Study.....	11
1.6 Overview of the Methodology .....	13
1.7 Thesis Overview .....	15
1.8 Summary .....	16
<b>Chapter 2: Literature Review</b> .....	<b>17</b>
2.1 Introduction .....	17
2.2 Factors that Affect Students’ Performance in Higher Education: Developing the Theoretical Framework.....	17
2.3 Factors That Affect Students’ Performance in Learning Introduction to Computer Programming.....	20
2.3.1 Prior computing experience.....	21
2.3.2 Year 12 performance in mathematics, physics and chemistry .....	22
2.3.3 Programming paradigms.....	23
2.3.4 First programming language.....	26
2.3.5 Programming environment .....	28
2.3.6 Teaching/learning methods and practices.....	30
2.3.7 Students’ ability in programming skills.....	33
2.3.8 Learning approaches .....	34
2.4 Measure of Student Success in CS1 .....	36
2.4.1 SOLO taxonomy .....	36
2.5 Theoretical Framework .....	42
2.6 Summary .....	44
<b>Chapter 3: Methodology</b> .....	<b>45</b>
3.1 Introduction .....	45
3.2 Preparation for Data Collection.....	45
3.2.1 Devise programming skills test questions .....	45
3.2.2 Design and select survey questionnaires .....	46
3.3 Research Participants .....	46
3.4 Research Methods .....	49

3.5 Research Questions .....	51
3.6 Data Collection Methods.....	52
3.6.1 Programming skills test .....	52
3.6.2 Survey questionnaire .....	53
3.6.3 Individual interviews .....	54
3.6.4 Group interviews .....	55
3.7 Classification of SOLO Levels.....	55
3.8 Ethical Considerations.....	77
3.9 Data Analysis .....	79
3.10 Summary .....	80
<b>Chapter 4: Students' Quantitative Univariate and Qualitative Results .....</b>	<b>81</b>
4.1 Introduction .....	81
4.2 Measure of Student Performance in CS1 .....	82
4.3 Student Quantitative Univariate Results .....	83
4.3.1 PST results .....	84
4.3.2 Quantitative survey results .....	87
4.4 Student Qualitative Results .....	102
4.4.1 Student qualitative survey results .....	102
4.4.2 Student interview results.....	105
4.5 Summary .....	116
<b>Chapter 5: Lecturer' Quantitative Univariate and Qualitative Results.....</b>	<b>119</b>
5.1 Introduction .....	119
5.2 Lecturers' Quantitative Univariate Results .....	119
5.2.1 Prior computing experience .....	120
5.2.2 Y12 performance in mathematics, physics and chemistry .....	121
5.2.3 Programming paradigm .....	122
5.2.4 Programming environments used in CS1 .....	122
5.2.5 First programming language to be taught in CS1 .....	124
5.2.6 Teaching/learning methods and practices that best suit students in learning CS1 .....	125
5.2.7 Lecturers' perceptions of the order of programming skills to be taught in CS1 .....	125
5.2.8 Lecturers' perception of the order of programming skills in terms of their contribution to the Student Performance in CS1 .....	126
5.3 Lecturer Qualitative Results .....	127
5.3.1 Lecturer qualitative survey results.....	127
5.3.2 Lecturer qualitative interview results .....	129
5.4 Inter-Rater Reliability.....	141
5.5 Summary .....	142
<b>Chapter 6: Bivariate and Multivariate Results.....</b>	<b>145</b>
6.1 Introduction .....	145
6.2 Student Bivariate Results .....	145
6.2.1 Association between student performance and students' prior computing experience.....	146
6.2.2 Association between student performance and students' Y12 performance in mathematics, physics and chemistry .....	147
6.2.3 Association between students' ability in programming skills .....	148
6.2.4 Association between student performance and students' ability in programming skills .....	149

6.2.5 Association between student performance and students' learning approach.....	150
6.3 Multivariate Results .....	152
6.3.1 Multiple regression analysis .....	152
6.3.2 Path analysis .....	157
6.4 Summary .....	161
<b>Chapter 7: Answers to Research Questions and Discussion .....</b>	<b>163</b>
7.1 Introduction .....	163
7.2 Answers to Research Questions and Discussion .....	163
7.2.1 What was the students' prior computing experience and does this affect performance in CS1? .....	163
7.2.2 What are students' and lecturers' experience/perceptions of first programming language, programming paradigm, programming environment and teaching/learning methods and practices? .....	165
7.2.3 What is the association, if any, between students' performance in CS1 and students' Y12 performance in mathematics, physics and chemistry? .....	169
7.2.4 What is the association, if any, between students' performance in CS1 and students' learning approach?.....	170
7.2.5 What is the association, if any, among the programming skill variables? .....	172
7.2.6 What is the association, if any, between students' performance in CS1 and students' ability in programming skills? .....	173
7.2.7 Is there a hierarchy among students' programming skills in terms of their contribution to student performance in CS1?.....	174
7.3 Suggestions on how to Improve Teaching/Learning of CS1 at RUB .....	176
7.4 Recommendations for CS1 at RUB.....	179
7.5 Summary .....	181
<b>Chapter 8: Conclusion .....</b>	<b>183</b>
8.1 Introduction .....	183
8.2 Research Findings .....	183
8.3 Wider Implications and Contribution .....	184
8.4 Limitations.....	186
8.5 Future Research Directions .....	188
8.6 Summary and Concluding Remarks .....	188
<b>Appendices .....</b>	<b>199</b>
Appendix A: Programming Skills Test Instrument.....	201
Appendix B: Programming Skills Test Marking Criteria Using SOLO Classification.....	205
Appendix C: Sample Students' Responses to Programming Skills Test Questions Marked Using SOLO Classification .....	217
Appendix D: Participant Consent Form .....	231
Appendix E: Participant Information Letter .....	233
Appendix F: Ethics Approval Letter .....	235
Appendix G: Student Survey Form.....	237
Appendix H: Lecturer Survey Form .....	241
Appendix I: Testing the Assumptions for Normality.....	245

Appendix J: Biggs Revised Study Process Questionnaire (R-SPQ-2F).....	253
Appendix K: Proposed CS1 Module Description .....	255
Appendix L: CS1 FSE paper of three colleges at RUB .....	259

## List of Figures

Figure 1.1. Example of algorithm design.....	5
Figure 1.2. Example of translating (a). ....	6
Figure 1.3. Example of translating (b). ....	6
Figure 1.4. Example of tracing.....	7
Figure 1.5. Example of explaining. ....	8
Figure 1.6. Example of writing. ....	8
Figure 1.7. Data collection methods. ....	14
Figure 2.1. Biggs 3P model of learning. ....	19
Figure 2.6. ‘Explain in plain English’ question analysed in Clear et al. (2008). ....	38
Figure 3.1. Individual components of an algorithm. ....	58
Figure 3.2. Suitable flowchart solution for algorithm design question 2.....	59
Figure 3.3. Individual components of flowchart. ....	60
Figure 3.4. R and RE SOLO levels and scores for algorithm design question 2 of the PST. ....	62
Figure 3.5. M and ME SOLO levels and scores for algorithm design question 2 of the PST.....	62
Figure 3.6. U and P SOLO levels and scores for algorithm design question 2 of the PST. ....	63
Figure 3.7. Suitable solution for translating question 1 of the PST. ....	64
Figure 3.8. Individual components of translating question 1 of the PST.....	65
Figure 3.9. R and M SOLO levels and scores for translating question 1 of the PST. ....	66
Figure 3.10. U and P SOLO levels and scores for translating question 1 of the PST. ....	67
Figure 3.11. R and M SOLO levels and scores for tracing question 2 of the PST. ...	68
Figure 3.12. U and P SOLO levels and scores for tracing question 2 of the PST. ....	69
Figure 3.13. R and RE SOLO levels and scores for explaining question 2 of the PST. ....	71
Figure 3.14. M and ME SOLO levels and scores for explaining question 2 of the PST. ....	71
Figure 3.15. U and P SOLO levels and scores for explaining Question 2 in a PST. ....	72
Figure 3.16. Individual components of writing question 2 of the PST. ....	73
Figure 3.17. R and RE SOLO levels and scores for writing question 2 in the PST... ..	75
Figure 3.18. M and ME SOLO levels and scores for writing question 2 of the PST. ....	75

Figure 3.19. U and UE SOLO levels and scores for writing Question 2 of the PST. ....	76
Figure 3.20. P SOLO levels and scores for writing Question 2 in the PST. ....	76
Figure 4.1. Number of student participants from seven programs in the survey. ....	87
Figure 4.2. Interface and screen of Turbo C++. ....	94
Figure 4.3. Interface of Dev-C++. ....	95
Figure 4.4. Command to compile and run C program using command line in windows. ....	96
Figure 4.5. Interface of Microsoft Visual Studio ....	96
Figure 6.1. Hypothesised path model. ....	158
Figure 6.2. Path diagram with FSE and writing as output variables and programming skills as predictor variables. ....	159
Figure 6.3. Path diagram with OS and writing as dependent variables and programming skills as independent variables. ....	160
Figure 6.4. Path diagram with MFSE and writing as dependent variables and programming skills as independent variables. ....	161
Figure 7.1. Approach of teaching/learning in CS1. ....	181



## List of Tables

Table 2.1 <i>SOLO Categories for Explaining Questions</i> .....	37
Table 2.2 <i>SOLO Categories for Code Writing Solutions</i> .....	41
Table 2.3 <i>SOLO Categories for Code Writing Solutions</i> .....	42
Table 3.1 <i>Number of Students Who Participated in a PST From Each Program</i> .....	48
Table 3.2 <i>Number of Students Who Participated in a Survey From Each Program</i> .....	48
Table 3.3 <i>Overview of Student Survey Questionnaire</i> .....	54
Table 3.4 <i>SOLO Levels for Algorithm Design Questions</i> .....	61
Table 3.5 <i>SOLO Levels for Translating Questions</i> .....	65
Table 3.6 <i>SOLO Levels for Tracing Questions</i> .....	68
Table 3.7 <i>SOLO Levels for Explaining Questions</i> .....	70
Table 3.8 <i>SOLO Classification for Writing Questions</i> .....	74
Table 4.1 <i>Descriptive Statistics of Student Performance (PST, FSE, OS)</i> .....	83
Table 4.2 <i>Student Representation in this Study from Three RUB Colleges</i> .....	84
Table 4.3 <i>College Summary Statistics in PST</i> .....	85
Table 4.4 <i>Programme Summary Statistics in PST</i> .....	86
Table 4.5 <i>Summary Statistics of the Programming Skills and Overall PST Score</i> ....	86
Table 4.6 <i>Results of t-Test and Descriptive Statistics of FSE, OS and PST by Prior Programming Experience</i> .....	89
Table 4.8 <i>Results of t-Test and Descriptive Statistics of FSE, OS and PST by Students' Y12 Performance in Physics for Categories A and B</i> .....	92
Table 4.9 <i>Results of t-Test and Descriptive Statistics of FSE, OS and Test by Students' Y12 Performance in Chemistry for Categories A and B</i> .....	93
Table 4.10 <i>Number of Students' Response to First Programming Language to be Taught in CS1 (n = 277)</i> .....	97
Table 4.11 <i>Teaching/Learning Methods (n = 277)</i> .....	99
Table 4.12 <i>Summary Statistics of the Order of Programming Skills to be Learnt in CS1 (n = 277)</i> .....	100
Table 4.13 <i>Summary Statistics of the Order of Programming Skills in Terms of Contribution to Student Performance in CS1 (n = 277)</i> .....	101
Table 4.14 <i>Aggregate Statistics on Students' Learning Approach Using Biggs Questionnaire (n = 277)</i> .....	102
Table 4.15 <i>Program and Gender of Student Participants in Individual Interviews</i>	105
Table 4.16 <i>College and Gender of Student Participants in Group Interviews</i> .....	106

Table 4.17 <i>Comparison of Students' Survey Open-Ended Question and Interview Responses to the Suggestion of How to Improve Teaching/Learning of CSI at RUB</i> .....	115
Table 5.1 <i>Lecturers' Response to Students' Prior Programming Language Experience (n=8)</i> .....	120
Table 5.2 <i>Lecturers' Responses to Prior Experience in Non-Programming Computer Activities (n=8)</i> .....	121
Table 5.3 <i>Summary Statistics of Students' Minimum Y12 Score in Mathematics, Physics and Chemistry, as Suggested by Lecturers</i> .....	121
Table 5.4 <i>Programming Environment Used by Lecturers to Teach CSI (n=8)</i> .....	122
Table 5.5 <i>Lecturers' Response to the First Programming Language to be Taught in CSI (n=8)</i> .....	124
Table 5.6 <i>Lecturer Response to Teaching/Learning Methods and Practices in CSI (n=8)</i> .....	125
Table 5.7 <i>Summary Statistics of the Order of Programming Skills to be Taught in CSI (n = 8)</i> .....	126
Table 5.8 <i>Summary Statistics on the Order of Programming Skills in Terms of Contribution to Student Performance in CSI (n = 8)</i> .....	127
Table 5.9 <i>Comparison of Student and Lecturer Suggestions on How to Improve Teaching/Learning of CSI at RUB</i> .....	129
Table 5.10 <i>Lecturer Interview Participants</i> .....	129
Table 5.11 <i>Comparison between Students' and Lecturers' Order of Programming Skills to be Learnt/Taught in CSI</i> .....	135
Table 6.1 <i>Association between Students' Prior Programming Experience and Student Performance (n = 327)</i> .....	146
Table 6.2 <i>Association between Students who Played Computer Games and Student Performance (n = 327)</i> .....	147
Table 6.3 <i>Association between Student Performance and Y12 Performance in Mathematics, Physics and Chemistry (n = 327)</i> .....	148
Table 6.4 <i>Association among Algorithm Design, Translating, Tracing, Explaining and Writing (n = 292)</i> .....	149
Table 6.5 <i>Association between Student Performance and Ability in Programming Skills (n = 327)</i> .....	150
Table 6.6 <i>Association between Student Performance and Students' Learning Approach</i> .....	151
(n = 327)	151
Table 6.7 <i>Verifying the Assumptions to Run Multiple Regression</i> .....	154
Table 6.8 <i>Multiple Regression Results with Student Performance as Dependent Variable and Programming Skills as Independent Variables</i> .....	154
Table 6.9 <i>Multiple Regression Results with Writing as Dependent Variable and Other Programming Skills as Independent Variables</i> .....	157
Table 7.1 <i>Researcher Proposed Outline of CSI</i> .....	176

## List of Abbreviations

CA	continuous assessment
CS1	Introduction to Computer Programming
CST	College of Science and Technology
FSE	final semester examination
GCIT	Gyalpoising College of Information Technology
GUI	graphical user interface
IDE	integrated development environment
IRR	inter-rater reliability
JNECS	Jigme Namgyel Engineering College
MFSE	modified final semester examination
OS	overall semester
PST	programming skills test
R-SPQ-2F	revised two-factor study process questionnaire
RUB	Royal University of Bhutan
SC	Sherubtse College
SOLO	structure of the observed learning outcome
Y12	Year 12



## Glossary of Terms

<b>Programming skills</b>	The basic essential skills required by students in Introduction to Computer Programming, such as algorithm design, translating, tracing, explaining and writing.
<b>Algorithm design</b>	Procedure to create a logic in solving a problem. In this study, algorithm design is represented in the form of algorithms and flowcharts. An algorithm is the process of writing detailed step-by-step instructions to solve a problem using either simple English or a pseudocode. A pseudocode is a mixture of English language and programming language constructs. A flowchart is a graphical representation of an algorithm.
<b>Translating</b>	The process of converting a given algorithm design into any high-level programming language.
<b>Tracing</b>	The process of manually executing program statements one by one, keeping track of values in all variables, updating the values as they change, and showing the desired output.
<b>Explaining</b>	The process of explaining in plain English the purpose of a given piece of code.
<b>Writing</b>	The process of writing executable code in any high-level programming language to solve a given problem.
<b>Student performance</b>	Students' scores in the programming skills test, final semester examination and overall semester performance.

# **Chapter 1: Introduction**

## **1.1 Introduction**

This thesis presents an investigation of factors that may affect the performance of students studying Introduction to Computer Programming (CS1). CS1 is the first, and compulsory, module studied by many undergraduate students from various fields of science, technology and engineering. Learning to program is not easy for many students upon their first introduction to it. This is because students not only have to learn to solve a problem, they also have to learn the syntax and semantics of a programming language to enable them to express their solution in a form that computers understand (Mannila, Peltomäki, & Salakoski, 2006). The difficulty faced by students in learning CS1 has led to high failure rates in CS1 taught at the Royal University of Bhutan (RUB). An important question behind the motivation for this study is: How can we improve student performance in CS1?

The following sections will present information on the thesis origin, background, research objectives and significance. An overview of the methodology and thesis concludes this chapter.

## **1.2 Thesis Origin**

This section details the origin of this thesis. It represents an opportunity to explore and research computer science education, particularly in the context of CS1. I began my teaching career at Sherubtse College (SC), RUB in 2008. Since then, I have taught computer science-related subjects to students enrolled in Bachelor of Science in Computer Science and Bachelor of Science in Physical Science.

The impetus for this study was my experience teaching CS1 to first-semester students and the similar experiences shared by other tutors teaching CS1 to science

and engineering students in other colleges of RUB. High failure rates in CS1 have been observed and identified as an issue.

When I interacted with students in my class, most indicate that learning to program is difficult since the content is new. This difficulty in learning to program is a challenge faced by many students at RUB when they are introduced to it in Semester 1.

As a tutor, when I reflected on my teaching experience and students' performance, I wondered whether it was the teaching methods/practices in the classroom or how students approached learning to program that determined their performance in CS1. With a full-time teaching load and other faculty responsibilities, research is near to impossible. However, I continued to persist in finding a solution to my research-related considerations and was exhilarated when I received a scholarship in 2015 to pursue my research interests at Curtin University, Western Australia.

### **1.3 Background**

This section presents the background information for this study by providing a brief history of RUB and the context of CS1 offered at RUB colleges. This will be followed by a clear definition of programming skills as used in this thesis.

RUB was established in 2003, offering tertiary education from various locations across the country (Department of Academic Affairs, October 2013). RUB is the only university in Bhutan (with 10 constituent colleges and two affiliated colleges under the umbrella of RUB) besides one medical university, Khesar Gyalpo University of Medical Sciences of Bhutan, which was established in 2013. Of 10 colleges, four offer degree and diploma programs that include CS1 in their first semester. All students taking those programs are required to take CS1. This mandatory module is the focus of this study.

The researcher has observed that students in these RUB programs have high failure rates in CS1 when they are introduced to it in their first semester. The researcher has experience as a participant observer and in teaching CS1 classes. During the assessment of students' programming solutions, the researcher observed a significant number of solutions that indicated that students were not achieving the intended course outcomes. The high failure rate and students' inability to comprehend the programming concepts at RUB is a cause for concern for the researcher and the institution. The researcher observed that of 60 students in one lecturer's classes, 50 percent failed (below 50 marks) in the semester-end results during spring 2015. A similar result was reported when the researcher taught the same course in spring 2013. The high failure rates in CS1 prompted the researcher to investigate the factors that may affect student performance in CS1.

Students' performance in CS1 was measured by the overall marks (out of 100) scored by students in a semester. It was based on students' performance in various assessment components: continuous assessment (CA) and final semester examination (FSE). CA constitutes 50 percent of the overall mark; it includes assignments, class tests, practical examinations and mid-semester examinations. The FSE is written at the end of the semester and constitutes 50 percent of the mark. These assessments test students' ability to explain the theoretical and practical components of programming.

Most students taking CS1 do not have any prior programming experience. The learning of CS1 takes place both in a classroom setting—a theory class—and in a computer laboratory—a practical class. Each theory class has 30–60 students and the practical class has 30 students maximum. Four hours a week are dedicated to theory (one hour per tutorial) and three consecutive hours a week comprise practical classes. The lecturer provides instructions in the theory class using slideshow presentations



and traditional teaching methods, such as notes on chalk and board or whiteboard. For the practical class, the lecturer uploads a tutorial/problem set in the learning management system in advance. Students then download these and begin planning and working on the problem during practical class. The lecturer assists students in understanding the problem and encourages them to work in pairs or groups. At RUB, a CS1 lecturer is responsible for not only delivering instructions, setting assignments, examination questions and laboratory problems and conducting laboratory sessions, but also for marking all assignments, examinations and laboratory problems. In Western universities, such as University of New Brunswick, Canada and Curtin University, Western Australia (researchers' observation when studying in these universities), teaching assistants and tutors assist in marking and conducting laboratory sessions. In one semester, students are required to submit several programming assignments, sit at least two examinations (mid-semester and final semester) and one practical examination in the computer laboratory. The assessments may differ slightly across colleges. For instance, the College of Science and Technology (CST) conducts three examinations in one semester.

In learning CS1, there are five basic but essential programming skills that every student is expected to master (Biró, Csenoch, Abari, & Máth, 2016; Crews & Ziegler, 1998; Hooshyar, Ahmad, Shamshirband, Yousefi, & Horng, 2015; Lister, Fidge, & Teague, 2009; Venables, Tan, & Lister, 2009). These skills are referred to in this thesis as 'programming skills' and they are algorithm design, translating, tracing, explaining and writing. This section will define these programming skills as understood by the researcher and to be used for the purpose of this study:

a) **Algorithm design** is a procedure to create a logic in solving a problem. This skill tests students' ability to analyse the problem, and understand and design a

complete solution. Algorithm design can be expressed in simple natural languages like English, pseudocodes or flowcharts (Goel, 2010; Hardnett, 2011). Both algorithms and flowcharts are used to represent algorithm design in this study. Algorithm refers to the process of writing detailed step-by-step instructions to solve a problem using either simple English or a pseudocode. A pseudocode is a mixture of English and the structural constructs of a programming language (Afriyie, 2007; Goel, 2010; Grover, 2001; Jeyapoovan, 2015). A flowchart is a graphical representation of an algorithm. An example of an algorithm design is shown in Figure 1.1.

Find the smallest number among the three numbers entered by the user and display the result.

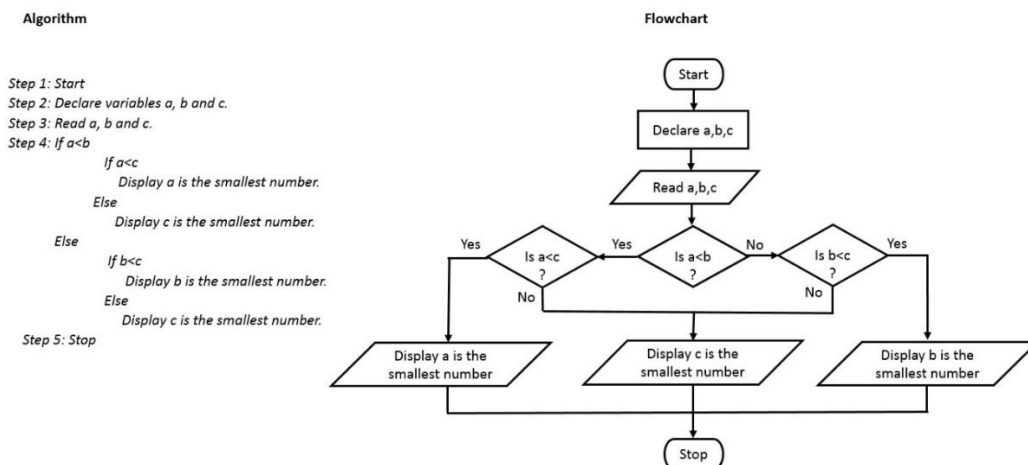


Figure 1.1. Example of algorithm design.

**b) Translating** is a process of converting a given algorithm design into any high-level programming language. The C programming language was used as an example in this study. This skill tests students' ability to correctly translate the logic from the given algorithm design into C programming language. This step requires students to know basic C programming language features. The translated program does not need to be an executable program, which means students are not required to remember the syntax, such as semicolons (;) double

quotation marks (“ ”) and commas (.). However, the basic programming features and logic must be correct, complete and almost close to an executable program. In fact, it is preferable if students can produce translated programs that are executable. Figure 1.2 and Figure 1.3 show examples of translating from an algorithm and flowchart to the C programming language respectively.

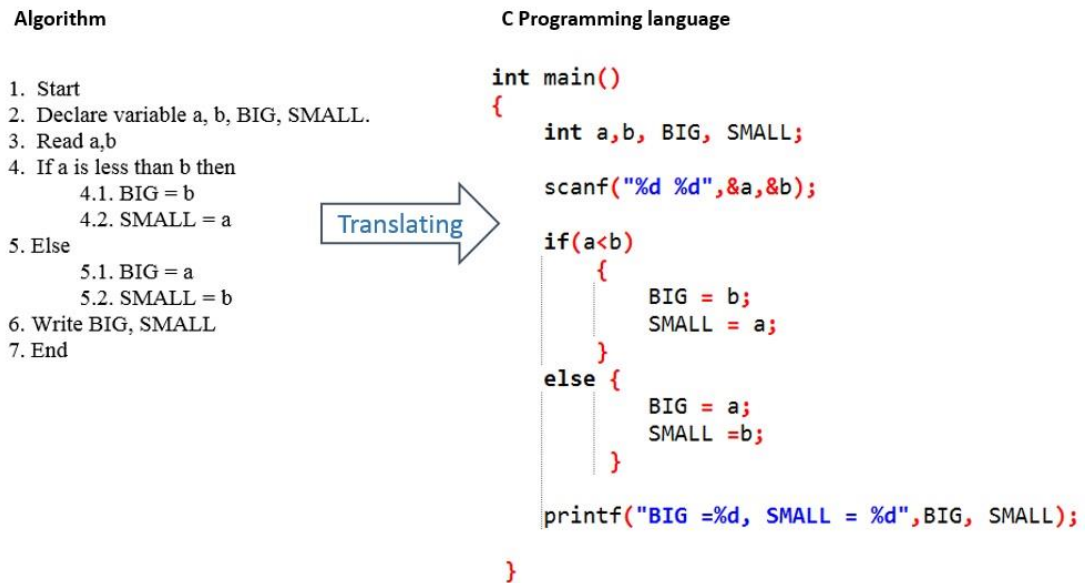


Figure 1.2. Example of translating (a).

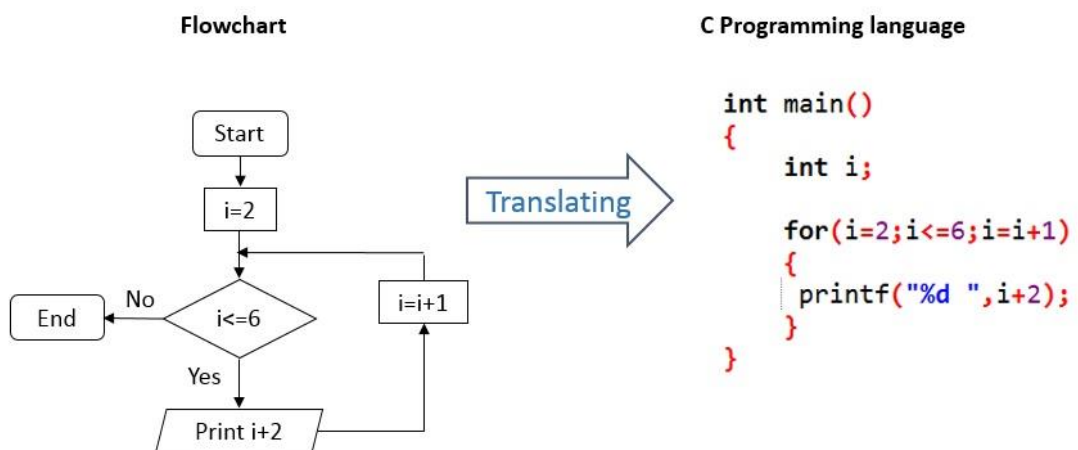


Figure 1.3. Example of translating (b).

c) **Tracing** is a process of manually executing program statements one by one, keeping track of the values in all variables, updating values as they change and showing the desired output. This skill tests students' ability to read a given piece of code and understand how the program works. An example of manually tracing a piece of code and producing a correct output is shown in Figure 1.4.

Study the given piece of code and answer the question that follows:

- Write the value of  $x$  when  $i = 1$
- Write the value of  $x$  at line 15.

Manually executing a program

	<code>int i, j, x = 0;</code>	<code>for (i = 0; i &lt; 3; ++i)</code>	<code>{</code>	<code>for (j = 0; j &lt; i; ++j)</code>	<code>{</code>	<code>x += (i + j - 1);</code>	<code>}</code>	<code>}</code>	<code>printf("x = %d", x);</code>
5	<code>int i, j, x = 0;</code>								
7		<code>for (i = 0; i &lt; 3; ++i)</code>							
8		<code>{</code>							
10			<code>for (j = 0; j &lt; i; ++j)</code>						
11			<code>{</code>						
12				<code>x += (i + j - 1);</code>					
13				<code>}</code>					
14				<code>}</code>					
15									<code>printf("x = %d", x);</code>

$i$	$i < 3?$	$j$	$j < i?$	$x = 0 + (i + j - 1)$	Print
0	$0 < 3$ (True)	0	$0 < 0$ (False)		
1	$1 < 3$ (True)	0	$0 < 1$ (True)	$x = 0 + (1 + 0 - 1) = 0$	0
		1	$1 < 1$ (False)		
2	$2 < 3$ (True)	0	$0 < 2$ (True)	$x = 0 + (2 + 0 - 1) = 1$	
		1	$1 < 2$ (True)	$x = 1 + (2 + 1 - 1) = 3$	
		2	$2 < 2$ (False)		
3	$3 < 3$ (False)				$x = 3$

Answer:

- $x = 0$
- $x = 3$

Figure 1.4. Example of tracing.

d) **Explaining** is the process of explaining, in plain English, the purpose of a given piece of code. This skill tests students' ability to comprehend the written code. Figure 1.5 shows an example of explaining.

Explain in plain English the purpose of the following code:

```
#include<stdio.h>
int main()
{
    int i,c=0;

    int a[10]= {2,3,4,5,7,8,9,12,34,6};

    for(i=0;i<10;i++)
    {
        if(a[i]%2==0)
        {
            c++;
        }
    }

    printf("%d",c);
    return 0;
}
```

Answer:

*The code counts the number of even numbers in an array of 10 elements.*

Figure 1.5. Example of explaining.

- e) **Writing** is the process of writing code in C programming language to solve a problem. This skill tests students' ability to write correct programs in C. In writing, students are expected to produce executable code that can run in a programming environment. An example of writing is as shown in Figure 1.6.

Write a program that will calculate the sum of every third integer, beginning with i=2 (i.e. calculate the sum of 2 + 5 + 8 + 11 + ..... ) for all values of i that are less than 30.

```
#include <stdio.h>

int main()
{
    int i,sum;

    sum=0;

    for(i=2;i<30;i=i+3)
    {
        sum+=i;
    }
    printf("The sum = %d", sum);

    return 0;
}
```

Figure 1.6. Example of writing.

## 1.4 Research Objectives

This section states the objectives and specifies the research questions to be addressed in this study. The main aim of this thesis is to investigate the factors that may affect students' performance in CS1 at RUB. These were investigated via seven objectives:

1. Discover if students' prior computing experience has any impact on their performance in CS1.
2. Explore the first programming language, programming paradigm, programming environment and teaching/learning methods and practices as experienced/perceived by lecturers and students in CS1 that may improve students' performance in CS1.
3. Test for associations between students' performance in CS1 and their Year 12 (Y12) performance in mathematics, physics and chemistry.
4. Investigate associations between students' performance in CS1 and students' learning approaches.
5. Discover any relationship among the programming skill variables.
6. Investigate associations between students' performance in CS1 and programming skill variables.
7. Identify if there is a learning hierarchy in programming skills or not.

These research objectives are addressed by asking seven research questions:

1. What is students' prior computing experience and does this affect performance in CS1?
2. What are students' and lecturers' experiences/perceptions of first programming language, programming paradigm, programming environment and teaching/learning methods and practices?

3. What is the association, if any, between students' performance in CS1 and students' Y12 performance in mathematics, physics and chemistry?
4. What is the association, if any, between students' performance in CS1 and students' learning approaches?
5. What is the association, if any, among programming skill variables?
6. What is the association, if any, between students' performance in CS1 and students' ability in programming skills?
7. Is there a hierarchy among students' programming skills in terms of their contribution to students' performance in CS1?

The above research questions were specifically selected based on the relevance and the need for teaching and learning CS1 at RUB. For instance, research question 1 and 3 were selected to discover if students' prior computing experience and Year 12 (Y12) performance in mathematics, physics and chemistry has any impact on their performance in CS1 or not because in Bhutan student results in these areas are used to select students in science and engineering programmes where CS1 is a mandatory module in their first semester. Research question 2 was selected to explore on the first programming language, programming paradigm, programming environment and teaching/learning methods and practices that may improve students' performance in CS1. This is because the researcher wants to explore more on these variables at the other colleges under RUB as Sherubtse College (where the researcher has taught) used C as the first programming language which is a procedural programming paradigm and most lectures used Turbo C++ and Dev-C++ as the programming environment. The answers to this research questions will give the complete knowledge on the first programming language, programming environments and teaching/learning methods

and practices that may improve student performance in CS1 at RUB. Similarly, research question 4 was selected to investigate if students' learning approaches has any impact on their performance in CS1 or not as it would be beneficial to know the approaches of learning that Bhutanese students take in learning CS1 and also to examine which learning approach has the impact on student performance in CS1. Furthermore, the research questions 5 and 6 was selected to discover if Bhutanese students' results exhibit any relationship among the programming skills variables and also between the programming skill variables and the performance in CS1. In the same line, research question 7 was selected to discover if Bhutanese students' results show any existence of the learning hierarchy in programming skills or not. The answers to these research questions may help the lectures and students at RUB to improve teaching and learning of CS1.

### **1.5 Significance of this Study**

This section addresses the reasons why this study is significant. The results from this study made the following significant contribution, not only to RUB but also to computing research worldwide.

This study provides lecturers of CS1 at RUB a deeper insight into first programming language, programming paradigm, programming environment and teaching/learning methods and practices. This is significant because it enables lecturers to better reflect on their current teaching methods and practices and make informed decisions. It also adds to and validates the existing body of computing research in the first programming paradigm and language choice to be used in CS1 and teaching/learning methods and practices that may improve students' performance in CS1.



This study has also examined student approaches to learning in CS1, which is associated with student performance in CS1. Thus, this study has contributed to the body of computing research by exploring quantitatively the relationship between students' learning approaches and their performance in CS1.

For RUB to produce competent graduates in science and technology, it is important to focus on the foundation module (CS1) as this will facilitate the successful completion of the program. Part of the original contribution of this study comes from the fact that it is conducted in Bhutan. A comprehensive literature review revealed no studies have examined the variables used in this study in the country of Bhutan. Thus, this study sample is unique and provided valuable data for other researchers with similar interests in the computing discipline.

The identification of five foundational programming skills—algorithm design, translating, tracing, explaining and writing—contributed to the body of knowledge in regard to programming skills. Although considerable research has been conducted in programming skills, such as tracing, explaining and writing (see Chapter 2), students' skills in algorithm design and translating have not been explored alongside tracing, explaining and writing (to the researcher's knowledge). Therefore, this study has made a unique contribution to the body of knowledge in CS1 programming skills. Subsequently, the structure of the observed learning outcome (SOLO) taxonomy was adapted to evaluate students' responses to programming skills test (PST) questions (see Section 3.2.1 and Appendix A). This has contributed new information in regard to the SOLO descriptions for each programming skill, and thus, provide avenues for comparison of results. Both of these contributions (programming skills and use of SOLO taxonomy) are significant to the CS1 teaching community because they enable a deeper understanding of the two concepts leading to better practice, especially in

terms of focus and sequence of teaching the programming skills in introductory computing.

## **1.6 Overview of the Methodology**

This section outlines the research methodology used in this study. This study used a convergent parallel mixed methods design (see Section 3.4, page 52) to collect data through both quantitative and qualitative methods, but mostly quantitative. Qualitative data were collected to confirm/disconfirm quantitative results and provide deeper understanding (Creswell, 2013, p. 213). The qualitative and quantitative data were collected via a variety of methods: survey questionnaires (for lecturers and students), lecturer and student individual interviews, student group interviews and student PST scores. The rationale for choosing this method was to best answer the research questions and triangulate the results. Figure 1.7 shows the methods of data collection used in this study.



Student individual interviews



Student individual interviews



Student group interviews



Student surveys



PST



Lecturer interviews

*Figure 1.7.* Data collection methods.

Students from three colleges who have completed a course in CS1 in their first semester at RUB in July–November 2016 participated in this study. A total of 292 students participated in the PST and 277 students participated in the survey.

This study achieved 100 percent participation from lecturers from three colleges. Eight lecturers participated in this study, of which six taught the module in 2016 and two in the previous years.

Students’ ability in programming skills was measured using the PST answer scripts of students’, using SOLO taxonomy (Biggs & Collis, 1982) . The SOLO

description used to evaluate the responses to algorithm design, translating, tracing explaining and writing questions are shown in Chapter 3.

Students' approaches to learning CS1 were identified using the Biggs revised two-factor study process questionnaire (R-SPQ-2F) which consists of 20 closed-response questions scored on a 5-point Likert scale (Biggs et al., 2001). Other information was collected from the survey and interviews.

Data analysis was completed using SPSS for quantitative data. Qualitative data were manually classified according to each research question. Other areas unrelated to research questions but of research interest are noted. Qualitative data were collected to cross-validate the quantitative data.

## **1.7 Thesis Overview**

This thesis consists of eight chapters and 12 appendices. This chapter has introduced the thesis by describing the origin and background information of this study. The research objectives and research questions have been specified. The significance of this study has been discussed and the research methodology briefly described. Finally, an overview of each chapter has been provided.

Chapter 2 reviews the extant literature, describing factors affecting students' performance in higher education with particular focus on CS1. The theoretical framework built based on the literature will be discussed.

Chapter 3 provides an in-depth description of the research methodology used in this study. The research questions will be outlined, sample and measures described and various data collection methods detailed.

Chapter 4 and 5 presents the results of the students' and lecturers' quantitative univariate analysis followed by qualitative results. Chapter 6 further presents the results of quantitative bivariate and multivariate analysis, including path analysis.

Chapter 7 discusses the results presented in Chapters 4, 5 and 6 by systematically answering each research question and discussing the answers in depth. Common key areas emerging from the qualitative data will be discussed.

Finally, Chapter 8 concludes this thesis with a brief summary of the findings with reference to the research questions proposed in this study. This chapter also outlines wider contributions and limitations and suggests future research directions.

Twelve appendices follow the references. Appendix A is a copy of the PST instrument. Appendix B is the PST marking criteria, using SOLO classification, while Appendix C is a copy of a sample student's responses to PST questions. Appendices D and E are copies of the participant consent form and information letter respectively. Appendix F is a copy of the ethics approval letter from the Human Research Ethics Committee of Curtin University. Appendices G and H are copies of the student survey and lecturer survey. Appendix I is the results of the assumptions for normality. Appendix J is the Biggs's revised two-factor study process questionnaire and Appendix K describes the proposed CS1 module. Finally, Appendix L is a copy of the FSE paper of three colleges at RUB.

## **1.8 Summary**

This chapter opened with an introduction to this thesis, outlining the sections addressed here. The sections described the origin of this thesis and background information related to this study. They stated research objectives and research questions, discussed the significance of this study, provided an overview of the methodology used in this study and an overview of each chapter. Chapter 2 will review the related literature.

## **Chapter 2: Literature Review**

### **2.1 Introduction**

The previous chapter introduced this study and how it originated. It provided background information, outlined research objectives and research questions and presented its significance and research methodologies, and described each chapter. This chapter will report on the literature that has explored the factors that affect students' performance in CS1.

This chapter begins with a review of the factors that affect students' performance in higher education, followed by an overview of the existing studies on the factors that affect student performance in the learning of units such as Introduction to Computer Programming (CS1), from which the theoretical framework was constructed. It then presents an overview of these factors listed under input and learning process factors. Further, the review of the measures of student success in CS1 is presented along with a review of the structure of the observed learning outcome (SOLO) taxonomy.

Overall, chapter 2 reviews the literature and in doing so, justifies the need for the current study.

### **2.2 Factors that Affect Students' Performance in Higher Education: Developing the Theoretical Framework**

It is often a challenge to measure students' academic performance in higher education, as it is dependent on different socioeconomic, psychological and environmental factors. Research has shown that socioeconomic factors such as gender, age, parents' income and educational background, alcohol consumption, substance abuse and high school performance can affect students' academic performance in

higher education (Hijaz & Naqvi, 2006; Mushtaq & Khan, 2012; Pritchard & Wilson, 2003).

The literature review identified that there is a significant influence on students' academic performance when students experience psychological factors such as stress, depression, anxiety or suicidal tendencies while pursuing higher education (Pritchard & Wilson, 2003). It was found that high level of stress affect students' mental, emotional and physical health, which might lead to depression and anxiety, thereby affecting students' ability to learn and potentially their academic performance (Kamtsios & Karagiannopoulou, 2015). The top three academic stressors identified were examination, excessive content to learn and lack of time to revise what has been learnt (Yusoff, Rahim, Baba, Ismail, & Pa, 2013). Rising expectations and responsibilities may generate negative emotions, which in extremes, can lead to suicide (Bhattacharya & Bhattacharya, 2015).

Environmental factors such as social support from peer and teacher, family encouragement and support, peer relationships, teacher–student relationships, experience of teachers and classroom facilities play a vital role in students' academic achievement. Social support is commonly defined as the existence or availability of people on whom we can rely, people who let us know they care for, value and love us. Social support has been identified as a resource that enables individuals to cope with stress (Yang, 2004). Family encouragement and support have been identified as important contributing factors for students' academic success (Gloria & Robinson Kurpius, 2001).

Biggs (1987) used the 3P model (see Figure 2.1) to describe the factors that may influence students' performance in learning. The Biggs 3P model comprises three components: presage, process and product. The presage component exists before the

student enters the learning context and includes factors such as prior knowledge, abilities, intelligence quotient and personality characteristics (e.g., age, sex and home background). It also includes situational factors such as subject area, teaching methods, time taken to complete a task and the structure of the course. The process component describes the learning situations and includes factors such as students' motives and approaches to learning. The product component is the outcome after learning has taken place. The outcome can be objective, for example, in terms of examination scores or subjective or the level of satisfaction attained. Biggs stated that product was significantly influenced by presage and process factors.

The model shows how student presage factors interact with situational factors during the learning process and lead to the achievement of the product. It argues that the instructor is responsible for the design and structure of the learning environment and the student is responsible for engaging appropriately with the activities.

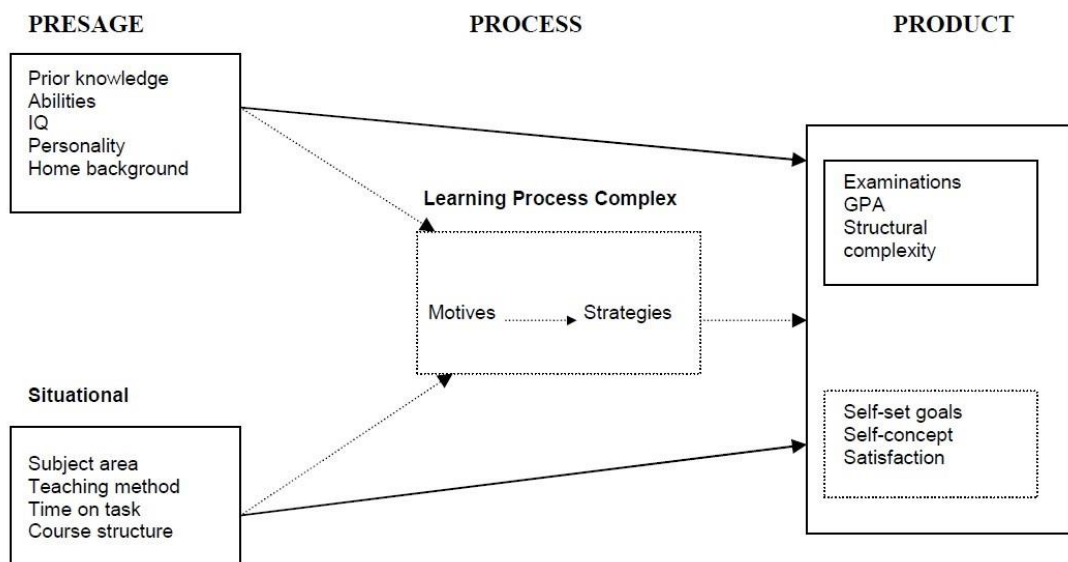


Figure 2.1. Biggs 3P model of learning.

For simplicity, this study has replaced the term presage with input factors, process with learning process factors and product with student performance. Section 2.3 provides an overview of the existing studies addressing the factors that affect



student performance in the learning of CS1, from which a more specific theoretical framework will be constructed and presented toward the end of this chapter.

## **2.3 Factors That Affect Students' Performance in Learning Introduction to Computer Programming**

Several factors that may influence students' performance in CS1 at university-level courses have been discussed in the literature. These factors are previous programming experience, previous non-programming computer experience, encouragement to pursue computer science, motivation, comfort level in the course, work style preference, attribution to success/failure, self-efficacy, mental model, learning approaches, gender, mathematics background (Bergin & Reilly, 2005a; Wiedenbeck, Labelle, & Kain, 2004), teaching methods (Krpan, Mladenović, & Rosić, 2015) and cognitive factors such as problem-solving, abstract reasoning, problem translation, skills, logical ability and cognitive style (Bergin & Reilly, 2005b). Other factors that may affect students' performance and are of research interest include students' ability in programming skills (Tan & Venables, 2010), programming paradigm (Gupta, 2004), programming environment and first programming language choice.

Wilson and Shrock (2001), Hagan and Markham (2000) and Wilson (2002) reported that comfort level in class was the best predictor of course success followed by mathematics background. Hagan and Markham (2000) and Kersteen, Linn, Clancy, and Hardyck (1988) found that previous programming experience and previous non-programming computer experience were an indicator of success in programming. Conversely, Bergin and Reilly (2005b) reported no significant difference between students with or without previous programming experience and between students with or without previous non-programming computer experience.

While considerable research has been done on factors that affect student performance in CS1, the parameters used were different. The literature review revealed that the parameters used were: students' enrolment program (computer science, social science, humanities and engineering), level (degree and diploma), educational settings (the United States, United Kingdom, India, Europe etc.), programming language taught (Python, Java and C++) and the measure of student success in CS1 (overall computer science grades, mid-term grade, final semester exam grade and continuous assessment). The parameters used in this study were based on situational factors currently evident at RUB. These were: students' enrolment in a computer science and engineering course at degree or diploma level, Bhutanese educational settings, teaching of C as the programming language, measurement of student success in CS1 as final semester examination (FSE), overall semester (OS) and programming skill test (PST) scores.

Sections 2.3.1–2.3.8 briefly review research studies for each of the factors that are likely to impact learning in CS1:

### **2.3.1 Prior computing experience**

Prior computing experience can be further categorised as prior programming experience and prior non-programming computer experience (Bergin & Reilly, 2005b; Wilson, 2002). Prior programming experience includes any prior formal programming or self-initiated programming courses taken outside a formal class. Prior non-programming computer experience includes using the Internet for information searches: games (online or offline) and application software such as Office, spreadsheets, presentation programs and databases (Wilson, 2002).

Studies have shown that students with prior programming experience performed significantly better in CS1 than those without (Hagan & Markham, 2000;

Holden & Weeden, 2003; Taylor & Luegina, 1991; Wilcox & Lionelle, 2018; Wilson, 2002). In Hagan and Markham (2000) and Holden and Weeden (2003), Java was the language used in their first programming course. They found a significant difference between students with prior experience in programming and those without. Moreover, the more programming languages in which students had experience, the better was their performance.

Few studies have examined the impact of prior non-programming computer experience on students' performance in CS1 (Bergin & Reilly, 2005b; Wilson, 2002). Wilson (2002) found that students with previous programming experience and game playing had significantly better or worse performance in CS1 respectively. Previous programming experience had a positive influence, while game playing had a negative influence on mid-term grade. Conversely, Bergin and Reilly (2005b) found no significant difference between students with or without previous programming experience or between students with or without non-programming computer experience.

Therefore, based on the literature, the researcher considered it important to examine these two variables in Bhutanese environment so the results from this study can better inform the Education Ministry of Bhutan on whether to include basic programming subjects in Year 11 and 12, or if not, for science students who may be likely to take CS1 in their first semester of university.

### **2.3.2 Year 12 performance in mathematics, physics and chemistry**

In computer programming, problem-solving can be broken down into four steps: 1) understand the problem, 2) determine how to solve the problem, 3) translate the solution into a computer language program, and 4) test and debug the program (Winslow, 1996). Since students learn general problem-solving skills in high school

mathematics, several studies have determined that student mathematics background is a good indicator of success in CS1 (Konvalina, Wileman, & Stephens, 1983; Patil & Goje, 2009; Wilson, 2002; Winslow, 1996; Zaffar, Hashmani, & Savita, 2018).

In Byrne and Lyons (2001), mathematics background was shown to have significant influence on programming performance. Byrne and Lyons (2001) and (Campbell & McCabe, 1984) beliefs that students' mathematics concepts to comprehend to master mathematics problems are similar to those of programming. They involve problem-solving ability, which is crucial for success in science and engineering. The researcher also observed while teaching CS1 that students who achieved high grades in Year 12 (Y12) mathematics perform better than those who did not achieve high grades. Similarly, prior performance in science subjects, although less studied, has shown to have a positive influence on students' performance in programming (Byrne & Lyons, 2001; Werth, 1986; Zaffar et al., 2018).

After the review of the literature, the researcher found no studies that have examined students' Y12 performance in mathematics, physics and chemistry in relation to their performance in CS1, especially in Bhutan. Therefore, the researcher considered it necessary to establish whether there is any link between students' Y12 performance in mathematics and science subjects and their success in CS1. This would allow more or less attention to be given to students studying CS1.

### **2.3.3 Programming paradigms**

A programming paradigm is an approach to programming a computer based on a mathematical theory or a coherent set of principles. Each paradigm supports a set of concepts that makes it ideal for a certain kind of problem (Van Roy, 2009). In simple terms, paradigm means 'a way of doing things or thinking about things' (Burton & Bruhn, 2003).

The choice of programming paradigm for beginners is one of the most debated topics in the literature (The Joint Task Force on Computing Curricula Association for Computing Machinery (ACM) IEEE Computer Society, 2013). Therefore, a brief review on the discussion of two popular programming paradigms suitable for CS1 are presented in Sections 2.3.3.1–2.3.3.2. The researcher acknowledges that there are more programming paradigms, which are beyond the scope of this study (Van Roy, 2009).

### ***2.3.3.1 Object-oriented paradigm***

The object-oriented programming paradigm uses abstraction in the form of classes and object to create real-world environments. Just as an object in the real world may be described by its attributes and behaviours, an object in object-oriented programming consists of a collection of variables (attributes) and procedures (behaviours) bundled permanently together (encapsulated) as a unit. Similar objects are regarded as special instances (modified copies) of a general class; the class is used as the template to make the objects. Objects become the basic programming unit, rather than procedures. A natural development of this picture of classes and objects is to view them as programs in their own right. That is, a class or its object copy is a self-contained collection of data and code that can function as a complete subprogram. Contained within each class or object are all the elementary building blocks of primitive data types and the procedures that act on them (Burton & Bruhn, 2003).

As object-oriented programming becomes increasingly popular in the workplace, many college and tertiary institution have adopted object-oriented programming in their first programming courses (Kölling & Rosenberg, 1996; Wiedenbeck, Ramalingam, Sarasamma, & Corritore, 1999). Moreover, object-oriented programming is believed to directly support many of the software engineering concepts that are difficult to convey in procedural programming such as code re-use,

encapsulation, incremental development, testing and program design (Decker & Hirshfield, 1994). However, introducing object-oriented programming for beginners remains difficult, as acknowledged by many researchers. Kruglyk and Lvov (2012) stated that it requires the knowledge of object-oriented programming almost immediately, making it difficult for beginners. Kölling (1999) and Kölling and Rosenberg (1996) stated that the lack of a truly object-oriented development environment has created difficulty in teaching object-oriented programming. The most commonly used languages in introducing object-oriented programming are C++ and Java.

### ***2.3.3.2 Procedural paradigm***

The procedural programming paradigm is a linear or top-down approach in programming. It uses procedures or subroutines to perform computations. According to Burton and Bruhn (2003), the main steps in writing a simple procedural program may be summarised as:

1. Read and understand the problem.
2. Devise a solution to the problem.
3. Formalise the solution as an algorithm.
4. Write the program.
5. Test and debug the program.
6. Document the program.

Burton and Bruhn (2003) asserted that students must master these steps before considering modelling a real-world object, which means students must first learn procedural programming and then move to object-oriented programming.

Gupta (2004) also recommended procedural programming for beginners, stating that procedural programming is easy to understand and continues to be the most

popular choice among teachers for an introductory programming module. He further added that procedural programming is ‘straightforward to convert intuitive algorithms into code’ while in object-oriented programming, one needs to ‘actually go into the structure and design of the system and how its components interact with each other’ (Gupta, 2004).

Although procedural programming has been taught in first programming courses for a long time, many universities are now slowly moving to object-oriented programming. The main reason for this transition is the ‘dread of paradigm shift’, which means once your mind is set on procedural programming, it takes time to switch to object-oriented programming. Thus, Kölling (1999) proposed to begin with object-oriented programming. He stated that the path to object-orientation through procedural is not required. Conversely, Brilliant and Wiseman (1996) reported that students find procedural programming easy to understand and find object-oriented programming difficult, so it is better to start with procedural programming. Therefore, this study will collect information from lecturers who have taught CS1 at RUB in regard to the first programming paradigm that better suits CS1. The most commonly used languages in introducing the procedural programming are Pascal and C.

#### **2.3.4 First programming language**

Choice of programming paradigm drives the choice of programming language. It is crucial to choose the suitable first programming language, as it will have a profound impact on ‘programming style, coding technique, and code quality’ (Gupta, 2004). Gupta (2004) discussed some of the requirements of a good introductory programming language. An introductory programming language for university students should be simple and easy to understand, orthogonal (i.e., not too many ways

of doing the same thing) and have simple input/output functions. Take, for instance, the most popular ‘Hello, World!’ program, written in Java as shown in Figure 2.2:

```
class HelloWorld
{
    public static void main(String args[])
    {
        System.out.println("Hello, World!");
    }
}
```

Figure 2.2. ‘Hello, World!’ program, written in Java.

In Prolog, it is written as shown in Figure 2.3:

```
hello :- printstring("Hello, World!").
printstring[].
printstring[H|T] :- put(H), printstring(T).
```

Figure 2.3. ‘Hello, World!’ program, written in Prolog.

In Python, it is written as shown in Figure 2.4 :

```
print('Hello, world!')
```

Figure 2.4. ‘Hello, World!’ program, written in Python.

And in C, it is written as shown in Figure 2.5:

```
int main()
{
    printf("Hello, World!");
}
```

Figure 2.5. ‘Hello, World!’ program, written in C.

To someone new to programming, the Python and C code look simple, easy and straightforward to understand. Gupta (2004) suggested C as the better choice for



new college students, as C is simple enough for students to be able to immediately write simple programs while being flexible enough to allow students to gradually learn to write complex programs. Moreover, he said that C introduces the basic elements common to most widely used ‘real-world’ programming languages, and provides a good foundation for learning other languages.

Other studies (Dierbach, 2014; Radenski, 2006; Sanders & Langford, 2008; Shannon, 2003; Yadin, 2011) also suggested Python as the better choice of programming language for CS1, as the syntax is simple and the structure is uncomplicated, which means it is easy for students to read and write code. Instructors also find it easy to teach Python, as the language is so simple that students do not create typical syntax errors due to missing semicolons or braces (Shannon, 2003). This frees students from detailed language syntax and allows them to concentrate on algorithms and problem-solving (Yadin, 2011).

Conversely, McCracken et al. (2001) reasoned that there is no difference between the programming languages; rather, it is a matter of how the course is taught. Therefore, this study will explore lecturer and student perceptions of the first programming language choice, as this issue has not been settled in the literature.

### **2.3.5 Programming environment**

Many programming environments can be used in learning to program (Gómez-Albarrán, 2005). One such environment is the integrated development environment (IDE). IDE comes packaged with a code editor, a compiler, a debugger and a graphical user interface (GUI) builder, which makes it easier for beginners to learn programming. Sophisticated programming environments are not suitable for beginners, as they assume a level of sophistication that beginning students do not possess. They are designed for experts, not students. These types of environments not

only waste students' time in learning to use the environment, but also increase students' level of frustration (Freund & Roberts, 1996; Kruglyk & Lvov, 2012). Thus, valuable time has to be spent teaching students to use the environment rather than teaching the concepts of programming. Moreover, the error messages generated by sophisticated environments are often uninformative, sometimes misleading and often require knowledge of advanced topics that novice students have yet to learn (Gómez-Albarrán, 2005). To address these drawbacks, research has focused on developing simple programming environments that students can easily use to write, compile, execute and debug their codes.

Freund and Roberts (1996) developed a programming environment called Thetis at Stanford University for students learning C in introductory computer science courses. Thetis consists of a C interpreter and associated user interface that provides students with simple and easily understood editing, debugging and visualisation capabilities. Both students and instructors have indicated that Thetis provides a better learning environment for students in CS1.

Lipman (2014) created CS1, a web-based programming environment for first-year computer science students to write, run and debug C programs. Students do not have to consider text editors or IDEs, Linux commands and compiling. Using CS1 has yielded good results in improving students' comprehension of CS1 concepts. AnimPascal, BlueJ and DrJava are other programming environment developed for Pascal and Java programming language (Gómez-Albarrán, 2005).

This issue of using suitable programming environments for CS1 students continues to be debated in the literature. Therefore, the perceptions of lecturers and students using programming environments adopted to learn CS1 using C at RUB will be investigated in this study.

### 2.3.6 Teaching/learning methods and practices

Since most students struggle to learn to program in their first semester, it is crucial to examine appropriate teaching/learning methods and practices that assist students in overcoming the difficulties of learning to program. Several teaching/learning methods and practices were reported in the review of the literature that may benefit student in learning to program: pair/group programming, live coding, game-based learning, puzzle-based learning, problem-based learning, pre-recorded lecturers and approach, deployment, result and improvement (ADRI).

**Pair programming** is a method in which two programmers work side by side on the same code at the same computer (Thomas, Ratcliffe, & Robertson, 2003). The two programmers take the role of a driver and a reviewer. The driver creates a code and takes control of the keyboard and mouse, while the reviewer reviews the code. They switch roles after a period (Bevan, Werner, & McDowell, 2002; McDowell, Werner, Bullock, & Fernald, 2002). Porter, Guzdial, McDowell, and Simon (2013), Thomas et al. (2003), McDowell et al. (2002), Bevan et al. (2002) and Williams, Wiebe, Yang, Ferzli, and Miller (2002) showed that students who used pair programming produced quality programs and learnt the materials faster. **Group programming** is similar to pair programming, but involves more than two students working on programming assignments. The only study in Bhutan reported that students learning programming in groups performed better in their examinations than those coding alone or watching someone else coding (Tshering, Lhamo, Yu, & Berglund, 2017). Also, in group programming, students are reported to be more motivated and gain experience in developing portions of a program in a group (Chamillard & Braun, 2000). Pair and group programming methods encourage students to collaborate and discuss with their peers which leads to better learning. Also,

programs are completed in less time, are better designed and have fewer errors. Students are also better motivated to stay on task, have more confidence in their solutions and show a positive attitude towards collaboration (Mohorovičić & Strčić, 2011).

**Live coding** involves solving programming problems by lecturers writing code from start to finish in the class during instruction periods (Paxton, 2002; Rubin, 2013). This method and practice in CS1 instruction requires lecturers to narrate their thoughts and actions while typing, compiling and testing code, and allowing students to participate in the coding process. Live coding can provide excellent learning opportunities for the students, such as code design and the debug process, which are very useful skill in programming. Rubin (2013) and Paxton (2002) found live coding to be an effective method in improving student learning outcomes over verbal explanation of static code examples.

**Game-based learning** teaches students the concepts of programming through understanding how a game works. Students reported that they had to spend more time understanding how a game worked than understanding the programming concepts. Once they understood, their motivation and enthusiasm increased (Mohorovičić & Strčić, 2011). Kazimoglu, Kiernan, Bacon, and Mackinnon (2012) designed an educational game called *Program your robot*, which enables students to practice working with introductory programming constructs within an environment that facilitates learning of introductory computer programming skills such as algorithm building, debugging and simulation. Students enjoyed playing the game and reported that this type of approach enhanced problem-solving abilities for those learning introductory computer programming. Other studies have incorporated games in

introductory computer programming, which increased students' motivation and achieved significant learning gains (Hicks, 2010; Sung et al., 2010).

**Puzzle-based learning** aims to teach students critical thinking and problem-solving techniques (Merrick, 2010). In puzzle-based learning, students reconstruct program pieces by selecting the correct program piece and placing them in the correct order (Mohorovičić & Strčić, 2011). Research has shown that puzzle-based learning increases students' interests in participating in programming courses (Merrick, 2010).

**Problem-based learning** engages students in problem-solving. Students can work individually or in groups and solve a problem by applying the knowledge they learnt earlier. Research has shown that problem-based learning has long-term benefits for students. Students can retain knowledge for longer and achieved better results in follow-up courses than did students who used traditional methods of learning. Problem-based learning can also enhance students' communication skills, creative thinking, motivation and responsibility (Mohorovičić & Strčić, 2011).

**Pre-recorded lectures** are multimedia recordings of lectures made available for students who are slow in absorbing information in class. Although pre-recorded lectures made no significant difference in students' final grades, students offered positive feedback and stated that pre-recorded lectures helped them to better understand some programming concepts.

**ADRI** is another approach of teaching and learning CS1 introduced by Malik and Coldwell-Neilson (2017). ADRI stands for approach, deployment, result and improvement. Four stages of ADRI were proposed by Malik and Coldwell-Neilson (2017) to enhance students' learning outcomes in CS1. The first stage (approach) covers problem-solving skills such as pseudocodes and flowcharts. The second stage (deployment) emphasises programming knowledge such as syntax and semantics of

the programming language. The third stage (result) deals with program input, the process used to solve a problem statement and expected outputs, while the fourth stage (improvement) provides more practice with different programming language constructs. The teaching/learning materials were prepared based on the four stages of the ADRI approach. An ADRI-based editor was also developed to support the ADRI approach and assist students in the learning process. Their study reported that ADRI was a better teaching/learning approach than the traditional approach, as ADRI provides all the basic skills required to comprehend programming constructs. Further, the final exam grades showed that the students performed better in the course offering ADRI than students who finished the course offered with the traditional approach.

Discussion with lecturers at RUB determined they currently practise pair/group programming and live coding when teaching CS1. Puzzle-based, game-based and problem-based learning is in-built with laboratory programming exercises done in pairs and programming assignments done in groups. Therefore, this research seeks to examine the benefits of practising pair/group programming and the live-coding method of teaching/learning in CS1 and further explore other teaching/learning methods and practices that have benefited students in learning to program. This can be gauged using surveys and interviews. Results are presented in Chapter 4 and 5.

### **2.3.7 Students' ability in programming skills**

An extensive study on the relationships between code tracing, explaining and writing skills was conducted by a multi-institutional, multi-national group in computer science education—the BRACElet Project (**B**uilding **R**esearch in Australasian **C**omputing **E**ducation) (Tan & Venables, 2010). Another BRACElet study by Philpott, Robbins, and Whalley (2007) found that students who could only trace code below 50 percent could not usually explain code. In the similar study, Sheard et al.

(2008) found that the ability of students to explain code correlated positively with their ability to write code. Studies within the BRACElet Project suggest the possibility of hierarchy in programming skills (Lister et al., 2010; Lister et al., 2009). First, the student acquires the ability to trace code. The ability to explain code develops, followed by the ability to write code when students become competent in both tracing and explaining (Lister et al., 2009; Schoeman, Gelderblom, & Smith, 2012; Venables et al., 2009). This hierarchy was further investigated in this study. However, according to Schoeman et al. (2012), the skills do not necessarily develop in strict hierarchical order. Instead, they may develop in parallel and reinforce each other.

Following the existing literature (Biró, Csenoch, Abari & Máth, 2016; Crews & Ziegler, 1998; Hooshyar, Ahmad, Shamshirband, Yousefi & Horng, 2015; Lister, Fidge & Teague, 2009; Venables, Tan & Lister, 2009) and based on researcher experience teaching CS1, the researcher identified five essential skills that students are expected to acquire when learning CS1. These skills are referred to in this study as ‘programming skills’ and they are algorithm design, translating, tracing, explaining and writing. Algorithm design and translating were added as unique elements to the study and were not mentioned in the literature as a whole. These additional elements were hypothesised based on the researcher’s personal experience teaching CS1 at RUB. It’s good to explore the relationship amongst the programming skills and also examine if there exist any hierarchy in the programming skills in Bhutanese environment. These programming skills are explained, with examples, in Section 1.3.

### **2.3.8 Learning approaches**

There are number of ways that students approach their studies (Buckley, Pitt, Norton, & Owens, 2010; Gadelrab, 2011; Schmeck, Geisler-Brenstein, & Cercy, 1991). Higher education research has identified students’ approaches to learning as

having a significant impact on their performance (Biggs, Kember, & Leung, 2001; de Raadt et al., 2005; Tait & Entwistle, 1996; Trigwell & Prosser, 1991). Thus, to achieve student success, educators need to understand student learning, in particular, how students set about their learning tasks, their intentions and strategies, and how these affect the quality of their performance (Byrne, Flood, & Willis, 2002, p. 19).

According to Biggs (1987), learning approach may be classified as either deep or surface. Learning to program is also affected by how students approach their learning: either through a deep and surface approach. The deep approach of learning seeks to understand a topic, while the surface approach refers to memorising a topic and reproducing materials. In learning programming, a surface approach can be used to memorise the syntax of the programming language, but a deep approach is essential to understand the logic of the program construction (de Raadt et al., 2005).

de Raadt et al. (2005) used Biggs's R-SPQ-2F to measure students' learning approaches to tasks in a programming course. They reported that the correlation between the elements of the Biggs questionnaire and students' final marks in an introductory computer programming course was stronger than cognitive task and prior experience. As per the recommendation of de Raadt et al. (2005) to include R-SPQ-2F to explain students' success in programming, the researcher has chosen R-SPQ-2F to examine students' approaches to learning in programming. However, the researcher acknowledges there are other methods of examining student approaches to learning.

It would be interesting to investigate how Bhutanese students approach their learning in CS1 and also to explore the relationship between their learning approach and their performance in CS1. Accordingly, the results can enable lectures and students to make informed practices in the students' approach to learning in CS1 at RUB.



## 2.4 Measure of Student Success in CS1

This section discusses the extant studies on measuring students' performance in CS1. Previous research has used final course and mid-term grades to measure students' success in CS1 (Bergin & Reilly, 2005a; Malik & Coldwell-Neilson, 2017; Wiedenbeck et al., 2004; Wilson & Shrock, 2001). Based on the knowledge gained from the extant studies, this study used PST, FSE and OS scores to measure student performance in CS1.

The PST was designed and devised by the researcher to explore the programming skills identified in this study (see Section 3.2.1 and Appendix A). Students' PST responses were evaluated and classified according to the structure of the observed learning outcomes (SOLO) taxonomy. The SOLO taxonomy was adapted to determine in-depth information about students' level of understanding. Section 2.4.1 will present a detailed description of the SOLO taxonomy.

### 2.4.1 SOLO taxonomy

The SOLO taxonomy proposed by Biggs and Collis (1982) is an educational taxonomy used to evaluate the learning outcomes of the learner. It can be adopted across different disciplines (Jimoyiannis, 2013). SOLO is a hierarchical structure comprising five major levels (Biggs & Collis, 1982):

1. **Prestructural [P]:** This is the lowest level in the SOLO category. The student response in this level demonstrates significant misconception or the response is totally wrong.
2. **Unistructural [U]:** The student response in this level demonstrates a correct grasp of only some part of the task.

3. **Multistructural [M]:** The student response in this level demonstrates an understanding of most of the parts of the tasks but fails to integrate the parts as a whole.
4. **Relational [R]:** The student response in this level demonstrates an understanding of the task as a single coherent whole.
5. **Extended abstract [EA]:** This is the highest level in the SOLO category. The student response in this level demonstrates an understanding beyond the scope of the task, thereby creating a new situation or knowledge.

The existing literature shows how SOLO has been used reliably to classify students' response to programming skills tasks, such as reading and writing (Jimoyiannis, 2013; Lister et al., 2010; Sheard et al., 2008; Shi, Cui, Zhang, & Sun, 2017; Shuhidan, Hamilton, & D'Souza, 2009). Thus, SOLO is a reliable taxonomy to evaluate students' response to the programming skills tasks listed in this study.

The SOLO categories of Clear et al. (2008) shown in Table 2.1 were adapted in this study to evaluate students' ability to explain a given piece of code.

Table 2.1

*SOLO Categories for Explaining Questions*

SOLO Category	Description
Relational [R]	Provides a summary of what the code does in terms of the code's purpose (The 'forest')
Relational error [RE]	Provides a summary of what the code does in terms of the code's purpose, but with some minor error
Multistructural [M]	A line-by-line description is provided of all the code (the individual 'trees')
Multistructural omission [MO]	A line-by-line description is provided for most of the code, but with some detail omitted

SOLO Category	Description
Multistructural error [ME]	A line-by-line description is provided for most of the code, but with some minor errors
Unistructural [U]	Provides a description for one portion of the code
Prestructural [P]	Substantially lacks knowledge of programming constructs or is unrelated to the question

Source: (Clear et al., 2008)

In Clear et al. (2008), some students' responses to 'explain in plain English what the given piece of code does' at their end of first programming paper examination were analysed according to the SOLO taxonomy (shown in Table 2.1). How students' responses to the 'explain in plain English' question (see Figure 2.6 below) is demonstrated here.

```

public double method10A(double[] aNumbers)
{
    double num = 0;

    for(int iLoop = 0; iLoop < iNumbers.length; iLoop++)
    {
        num += aNumbers[iLoop];
    }
    return num;
}

```

*Figure 2.6.* 'Explain in plain English' question analysed in Clear et al. (2008).

Students who provided a summary of the purpose of the piece of code were assigned R (relational). Those who provided a summary with some minor errors were assigned RE (relational error), while those who provided a line-by-line description of all the code were assigned M (multistructural). Students who provided a line-by-line description for most of the code but with some details omitted were assigned MO (multistructural omission), while those who provided a line-by-line description for most of the code but with some minor errors were assigned ME (multistructural error). Finally, those who provided only a description for one portion of the code were assigned U (unistructural) and those who provided a response unrelated to the question

were assigned P (prestructural). The sample of students' responses and SOLO categories were:

- This method returns the sum of the numbers in the array **[R]**.
- Trying to add all the numbers stored in the arrayList that is less than the length of the arrayList. Go through each number from index 0 to the index just before, then end of the arrayList **[RE]**.
- It sets the double num to zero and executes the loop as long as iLoop is less than the length of aNumbers. If it does execute the loop, it adds the value of num to aNumbers and re-executes the loop by incrementing. Once the condition is no longer met, it returns num **[M]**.
- The method loops through the aNumbers and adds the aNumber to num and gives the output of that equation  $num + = aNumber[iLoop]$  **[MO]**.
- The method increments the loop and returns a number of type double as long as the loop is less than the length of aNumbers **[U]**.

The SOLO descriptions for the explaining questions adapted in this study are further explained in Chapter 3. Similarly, the SOLO categories for code writing tasks in this study were based on the SOLO categories proposed by Shuhidan et al. (2009); Whalley, Clear, Robbins, and Thompson (2011) and Lister et al. (2010).

According to Shuhidan et al. (2009) (see Table 2.2), each SOLO level represents increasing cognitive load. The lowest level, prestructural, represent students whose responses contained several pieces of unconnected information, but who were unable to connect as a whole. As the level increases, students' responses make more sense and make the necessary connections. Further, Shuhidan et al. (2009) found that it is important to analyse each component part of the solution and determine if students can link all components to fulfil the relational level. For example, the list of

components to satisfy for the writing question (i.e., write code to calculate the highest and lowest integer, from a set of integers passed via the commandline) would be:

- ability to create a loop
- ability to extract or convert the argument correctly
- ability to find the highest value
- ability to find the lowest value
- ability to code correctly.

This approach of listing components of the solution was used in this study and is explained in detail in Chapter 3.

Table 2.2

*SOLO Categories for Code Writing Solutions*

SOLO Category	Description
Extended abstract	Novices able to make connections beyond the scope of question and able to transfer knowledge a new situation
Relational	Fully correct or almost right. Novices appreciate significance in relation to the whole program and can generalise outside of program
Multistructural	Numbers of connections made. Novices can create code for loops and comparisons, but with a few minor slips, leading to failure to connect the whole idea. They may fail to convert arguments, use correct operators or interpret general explanation
Unistructural	Simple connections are made. Novices can compare or write loops but fail to implement or derive the connections of loops in relation to manipulation of arrays or usage of further structures
Prestructural	There are bits of unconnected information. Novices know something, but the overall argument makes no sense
No attempt or totally wrong	The answer is blank or totally wrong

Source: (Shuhidan et al., 2009).

According to Whalley et al. (2011) (see

Table 2.3), SOLO levels are placed into two phases, suggesting that learning passes through various stages from a more quantitative phase (surface) to a more qualitative phase (deep, connecting and relating ideas) as learning tasks and their complexity increase.

Table 2.3

*SOLO Categories for Code Writing Solutions*

Phase	SOLO Category	Description
Qualitative	Extended abstract— extending [EA]	Used constructs and concepts beyond those required in the exercise to provide an improved solution
	Relational— encompassing [R]	Provided a valid well-structured program that removes all redundancy and has a clear logical structure. Specifications were integrated to form a logical whole
	Multistructural— refinement [M]	Represented a translation close to a direct translation. Code may have been reordered to make a more integrated and/or valid solution
Quantitative	Unistructural— direct translation [U]	Represented a direct translation of the specifications. Code will be in the sequence of the specifications
	Prestructural [P]	Substantially lacked knowledge of programming constructs or answer was unrelated to the question

Source: Whalley et al. (2011).

The SOLO descriptions of Shuhidan et al. (2009) and Whalley et al. (2011) were used as a guide to develop the SOLO descriptions that fit this study in classifying student responses to writing questions. Chapter 3 presents the details of how the SOLO descriptions were adapted based on the literature.

## **2.5 Theoretical Framework**

The theoretical framework that underpins this study was Biggs 3P model of learning (Biggs, 1987). This model was chosen, as it fitted well with a study taking place in a university that examines factors that might affect students' performance in CS1. This study focused on the factors governing input and learning process factors,

and examined the relationships among those factors to improve students' performance in CS1.

The theoretical framework shown in Figure 2.7 was built based on the theoretical framework of the Biggs 3P model of learning. The specific factors identified for this study are based on the existing literature, the researcher's experience and discussion with the lecturers of RUB who are responsible for students' performance in CS1 at RUB.

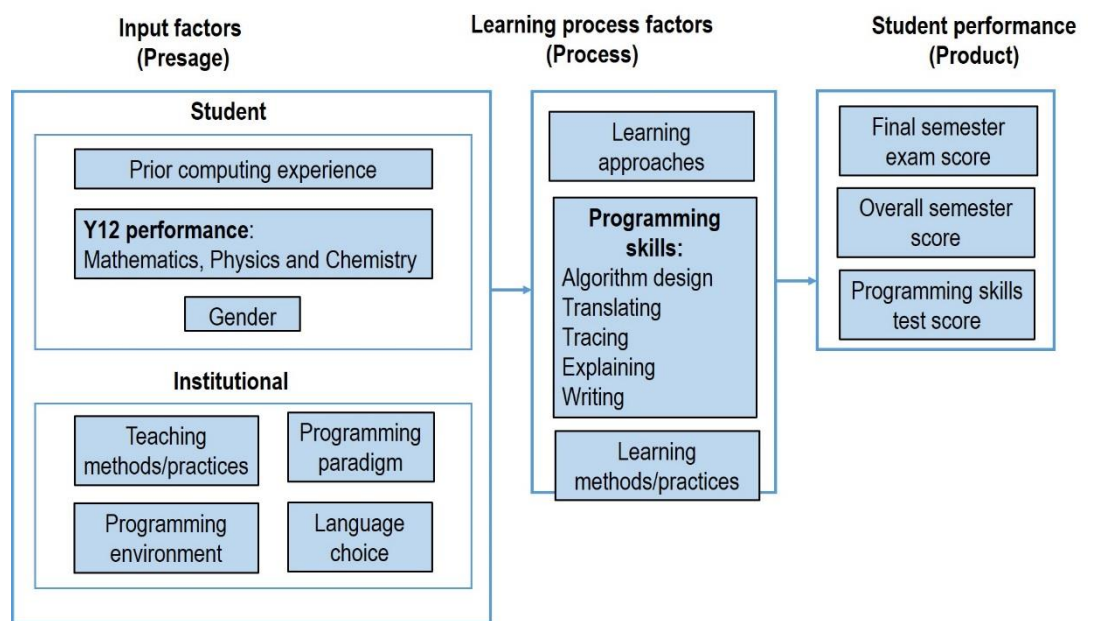


Figure 2.7. Theoretical framework

**Input factors** are those factors that students possess before entering the learning situation and commencing the study of CS1. Input factors identified in this study are prior computing experience, Y12 performance in mathematics, physics and chemistry, and gender. Input factors can also include institutional factors already in place, such as teaching methods and practices, programming paradigm, programming environment and language used in CS1.

**Learning process factors** are those factors that describe the strategies and activities while learning is taking place. Learning process factors examined in this



study are students' approaches to learning, ability in programming skills and learning methods and practices.

**Student performance** is the outcome of learning. Student performance in this study measures the achievement attained after a semester course in CS1. This achievement was measured in terms of their score in the FSE, PST and OS performance in CS1.

This study hypothesised that there is a relationships among input and learning process factors and student performance, as well as among the factors within these categories.

## **2.6 Summary**

In this chapter, the review of the literature relating to the factors that affect student performance in CS1 was conducted. The theoretical framework for this study was presented, along with the review of each of the elements listed in the conceptual framework. The review of the literature outlined the few studies that have examined the factors that affect students' performance in CS1. However, the parameters used in each study were different. Thus, this study will examine the factors that affect students' performance in CS1 for students enrolled in the computer science and engineering program at the degree and diploma level at RUB in the Bhutanese educational context. The use of SOLO taxonomy to analyse student responses to a question was also reviewed.

Chapter 3 will present a detailed description of the research methodologies used in this study. This includes preparation for the study, description of the data collection methods and overview of the data analysis.

## **Chapter 3: Methodology**

### **3.1 Introduction**

Chapter 2 provided a review of the literature on the factors that affect students' performance in CS1 and indicated a conceptual framework upon which studies of factors have been based. This chapter presents a detailed description of the methodology used in this study. It begins by describing the preparations carried out prior to data collection and recruitment of research participants. It explains why a mixed methods design approach was selected for this study, revisits the research questions and specifies unique variables of each research question. Data collection methods employed to gather data related to these variables are discussed next. It then explains the data analysis measures used to analyse quantitative and qualitative data.

### **3.2 Preparation for Data Collection**

Prior to data collection, preparations were made considering the key areas of this research. This section will describe these preparations.

#### **3.2.1 Devise programming skills test questions**

The programming skills test (PST) questions were designed by the researcher by examining the course syllabus and consulting with the lecturers from respective colleges. The questions were devised to cover the five categories of programming skills based on researcher experience and the earlier work of Lopez, Whalley, Robbins, and Lister (2008) and Lister et al. (2009). Two questions addressed each programming skill: algorithm design, translating, tracing, explaining and writing. There are 10 questions in total in the PST. Appendix A provides a copy of the PST instrument used for this study.

### **3.2.2 Design and select survey questionnaires**

In consultation with supervisors, the student survey questionnaire with open-ended questions was specifically designed for this study to address the variables in each research question. Similarly, the lecturer survey questionnaire was also designed specifically for this study to address similar variables. Appendices G and H include a copy of the student and lecturer survey questionnaires respectively.

To determine students' learning approaches in CS1, the Biggs's R-SPQ-2F, which consists of 20 closed-response questions scored on a 5-point Likert scale (see Appendix J), was selected to measure students' learning approaches (Biggs et al., 2001).

The student and lecturer survey questionnaires, consent forms (see Appendix D) and information letters (see Appendix E) were printed, packed and carried to Bhutan in February 2017.

### **3.3 Research Participants**

Research participants for this study were students enrolled in programmes in which CS1 is a compulsory module in the first semester at three constituent colleges of RUB. These three constituent colleges are Sherubtse College (SC), Jigme Namgyel Engineering College (JNEC) and College of Science and Technology (CST), which are geographically spread across the country and offer a variety of undergraduate programs.

SC offers programmes in different areas such as geography, economics, history, Dzongkha, population and development studies, political science, sociology, environmental science, media studies, English, life sciences, mathematics, chemistry, physics and computer science. Students enrolled in computer science were participants in this study.

JNEC offers programmes at both bachelor and diploma level, including power engineering, civil engineering, electrical engineering, mechanical engineering, computer hardware and networking, electronics and communication engineering, surveying, and materials and procurement management. Students enrolled in the diploma programme in computer hardware and networking were participants in this study.

CST offers programmes in civil engineering, electrical engineering, electronics and communications engineering, architecture and information technology. Students enrolled in all these programme were participants in this study.

The institution that provided the highest number of student participants was from the CST. This is because CST offers five programs in which CS1 is studied in the first semester, while JNEC and SC offer only one program.

The PST data were collected in November–December 2016 (i.e., end of the first semester). Other data, like survey and interviews, were collected in February–March 2017 (i.e., beginning of second semester). This study aimed to obtain 100 percent student participants, so no sampling technique was used.

The data were collected and entered into SPSS. The final sample consisted of 292 students in a PST of 342 students enrolled in July 2016, and 277 students in a survey of 309 students (some students failed in the first semester). Table 3.1 shows the number of students enrolled in the program and who participated in a PST. Table 3.2 shows the number of students (program-wise) in the second semester (total column) and those who participated in the survey.

Table 3.1

*Number of Students Who Participated in a PST From Each Program*

Program	Enrolled	Participated
B.Sc. Computer Science	27	26
Diploma in Computer Hardware and Networking	43	43
Bachelor of Architecture	19	11
B.Eng. in Information Technology	38	33
B.Eng. in Civil	116	98
B.Eng. in Electrical	68	54
B.Eng. in Electronics and Communication	31	27
Total	342	292

Table 3.2

*Number of Students Who Participated in a Survey From Each Program*

Programme	Total	Participated
B.Sc. Computer Science	27	27
Diploma in Computer Hardware and Networking	41	37
Bachelor of Architecture	19	18
B.Eng. in Information Technology	31	30
B.Eng. in Civil	109	83
B.Eng. in Electrical	51	53
B.Eng. in Electronics and Communication	31	29
Total	309	277

Eight lecturers who taught CS1 in 2016 and the previous year participated in a survey and an interview, representing 100 percent lecturer participation.

### **3.4 Research Methods**

As mentioned in Section 3.6, this study used various data collection methods such as a test, survey questionnaire with closed and open-ended questions, and individual and group interviews. These data collection methods involved both quantitative and qualitative data, but mostly quantitative. Creswell (2014) defined mixed methods design as the methods that ‘involves combining or integration of qualitative and quantitative research and data in the research study’. Creswell (2014) stated the existence of many designs in the mixed methods field, such as convergent parallel mixed methods, explanatory sequential mixed methods and exploratory sequential mixed methods.

The rationale for choosing a mixed method was to best answer the research questions listed in Section 3.5. The multiple quantitative and qualitative data collection methods were used in this study to answer the research questions. The quantitative data collected from the PST were used to measure students’ success across five programming skills. The quantitative and qualitative data collected from the student/lecturer survey questionnaire (which includes both closed and open-ended responses) was used to answer the research questions in regard to students’ prior computing experience, students’ and lecturers’ experience/perceptions of first programming language, programming paradigm, programming environment and teaching/learning methods and practices, students’ Y12 performance in mathematics, physics and chemistry, and students’ learning approach. The qualitative data from the semi-structured student/lecturer interviews and student group interviews were collected using the same variables to provide a deeper understanding of the quantitative findings and to triangulate the results (Creswell, 2013). Triangulation is

described as the use of two or more methods of data collection in the study of some aspect of human behaviour (Cohen, 2017).

The quantitative data were used to generalise the statistical tests to a larger sample and the qualitative data were used to achieve an in-depth analysis. The PST and quantitative and qualitative data from the closed and open-ended questions from the survey, and the qualitative data from interviews were collected independently at the same time. The qualitative data obtained from the quantitative data in the survey were analysed together, interpreted and reported under the quantitative survey results section in Chapter 4 and 5. For example, the survey question included: ‘In your experience and opinion, which programming language should you learn in CS1?’ Students selected one programming language from the list given (quantitative response). The next question was, ‘What made you choose that language?’ (qualitative response). The open-ended questions (i.e., ‘Do you have any suggestions on how we can improve teaching and learning of CS1?’) were analysed separately by classifying the responses based on the number of occurrences. These are reported under the qualitative results section in Chapter 4 and 5. The qualitative data from interviews were analysed separately and reported in the qualitative results section in Chapter 4 and 5. The quantitative and qualitative results were compared and contrasted in Chapters 4, 5 and 6. The qualitative results were used to confirm or disconfirm the quantitative findings.

The mixed methods design used in this study was convergent parallel mixed methods design (Creswell, 2014). According to Creswell (2014), convergent parallel mixed methods design involves both quantitative and qualitative data collected independently and in parallel to each other. They are analysed separately and converged to compare and contrast the findings.

### 3.5 Research Questions

As mentioned in Chapter 1, this study aims to address seven research questions:

1. What are the students' prior computing experience and does this affect performance in CS1?
2. What are the students' and lecturers' experience/perceptions of first programming language, programming paradigm, programming environment and teaching/learning methods and practices?
3. What is the association, if any, between students' performance in CS1 and students' Y12 performance in mathematics, physics and chemistry?
4. What is the association, if any, between students' performance in CS1 and students' learning approach?
5. What is the association, if any, among the programming skills variables?
6. What is the association, if any, between students' performance in CS1 and students' ability in programming skills?
7. Is there a hierarchy among students' programming skills in terms of their contribution to student performance in CS1?

The key variables in these research questions were students' performance, students' learning approaches, students' programming skills, students' prior computing experience, students' Y12 performance in mathematics, physics and chemistry modules, and students' and lecturers' experience/perceptions of first programming language, programming paradigm, programming environment and teaching/ learning methods and practices. To obtain a significant amount of data relating to each of these specific variables, this study employed a number of data collection methods, which will be discussed in Section 3.6.



## **3.6 Data Collection Methods**

This section will discuss the methods of data collection used in this study: PST, survey questionnaires, individual and group interviews. The survey was administered by the researcher after explaining the information sheets and obtaining consent to participants. Verbal consent was obtained for individual interviews and group interviews.

### **3.6.1 Programming skills test**

As mentioned in Chapter 1 and Section 3.3, due to the geographic location of the three participating colleges, the researcher personally was not able to administer all methods of data collection in the given time frame. Lecturers who were teaching CS1 were invited to assist the researcher in administering the PST after students had completed one semester of instruction in CS1. The information letter was sent via email to lecturers and the lecturers who volunteered provided their consent via email. The researcher then emailed consent forms, information letters and PST questions along with instructions on how to administer the PST.

The lecturers in their respective colleges printed and photocopied PST questions, consent forms and information letters to be given to students during administration of PST. This enabled administration of the PST in the same time frame in all three participating colleges. The researcher requested that all volunteer lecturers keep PST answer scripts in a safe place, which was collected after two months (i.e., after winter break) when the researcher visited the colleges for survey and interviews.

Subsequently, at Curtin University, the SOLO levels were initially classified by marking 30 sample PST answer scripts that were chosen randomly. The SOLO levels were gradually refined as necessary. The raw score of the equivalent SOLO levels were used as data for analysis. The details of how the SOLO levels were

classified and applied to evaluate students' responses to PST questions are shown in Section 3.7.

### **3.6.2 Survey questionnaire**

The student survey questionnaire was used to collect data on students' experience in programming skills, prior computing experience, Y12 performance in mathematics, physics and chemistry modules and experience/perceptions of first programming language, programming paradigm, programming environment and learning methods and practices. The students' overall semester (OS) score in CS1 was also recorded in the survey. The lecturer survey questionnaire was used to collect data on the same variables.

To collect data to determine student's learning approaches in CS1, the Biggs R-SPQ-2F, which consists of 20 closed-response questions scored on a 5-point Likert scale (see Appendix J), was used to measure students' learning approaches (Biggs et al., 2001). For example, the questions include:

Question 4: I work hard at my studies because I find the material interesting.

Question 7: I find most new topics interesting and often spend extra time trying to obtain more information about them.

Students' responses to questions are 'This item is...':

- A (scoring 1 point)—never or only rarely true of me
- B (scoring 2 points)—sometimes true of me
- C (scoring 3 points)—true of me about half the time
- D (scoring 4 points)—frequently true of me
- F (scoring 5 points)—always or almost always true of me.

A score for the deep approach was constructed by summing the deep motive and deep strategy subscales, and a score for the surface approach was constructed by summing the surface motive and surface strategy.

These survey questionnaires were administered in English by the researcher to volunteer students and lecturers of the three participating colleges. Table 3.3 provides an overview of the student survey instrument.

Table 3.3

*Overview of Student Survey Questionnaire*

Variables	Item No.	Data Type
Prior computing experience	1–3	Quantitative
Y12 performance in mathematics, physics and chemistry modules	4	Quantitative
OS score in CS1 (student performance)	5	Quantitative
Experience/perceptions of first programming language, programming paradigm, programming environment and student learning methods/practices	6–11	Quantitative+ Qualitative
Experience on programming skills	12–13	Quantitative
Open question on how we can improve teaching and learning of CS1	14	Qualitative
Learning approaches	15 (1–20)	Quantitative

### 3.6.3 Individual interviews

Student participants for individual interviews were selected from each program based on their FSE score. As far as possible, one student of the high performers, one of low performers and one of average performers were selected, ensuring the inclusion of at least one female student for gender equality. In some cases, students identified did not want to participate, which was respected. Instead, students from the same program were invited to volunteer. Verbal consent was then obtained from both

students and lecturers. A semi-structured face-to-face interview was conducted with the students and lecturers to collect data on the same variables (mentioned above in the survey questionnaire). This was done to support the quantitative data and explore key areas for this study. Results of individual interviews are presented in Chapter 4 and 5.

### **3.6.4 Group interviews**

Group interviews were conducted for each program. Interviews were semi-structured as an individual interview discussed above and directs a key areas for the group to discuss. The key advantage of group interviews is to bring together a group of students enrolled in the same program with varied opinions to generate a wider range of responses than in individual interviews (Cohen, 2017). The results of group interviews are presented in Chapter 4.

## **3.7 Classification of SOLO Levels**

This sections describes how students' PST responses were classified based on SOLO levels. When classifying according to the SOLO taxonomy, students' solutions were examined as a coherent whole (Shuhidan et al., 2009). Individual components were specified for each question under algorithm design, translating and writing skills (Shuhidan et al., 2009). These components are the building blocks that students must complete in response to a question. Those individual components were further classified with symbols: easier (E), harder (H), specified (S) and unspecified (U). The symbols E and H on each component were determined after marking 30 sample answer scripts based on the responses of students in PST for that particular question. For instance, if more than 50 percent of the sample students' responses missed to **typecast** the result of an integer division, this component of the question was assigned H, otherwise it was assigned E. Components with the symbol E were basic, while

components with the symbol H were harder and required more time and consideration than those marked E. This dichotomous approach was taken to avoid having to classify these extra components on a scale of difficulty. It was simplest to consider them as either easier or harder. The components with the symbol S were specified/given in the question and the components with the symbol U were not given in the question but were nonetheless essential to achieve a classification at the relational level. For the student to achieve relational level of understanding to a question, the response to that question had to demonstrate an ability to complete all components of the questions.

Based on the specification and the required complexity of the response to a question, the SOLO levels for each of the programming skills are different. For instance, algorithm design and explaining has five SOLO levels. This is because the student needs to spend some time in completing these tasks as well as the amount of tasks that needs to be completed is little more than translating and tracing skills. While in translating and tracing, it has three SOLO levels as the amount of effort and cognitive skills required here is not much compared to other programming skills since in translating, the algorithm and flowchart is already given and in tracing, the piece of program is already given as well. Writing has the highest levels of SOLO. This is because, students have to spent time in building the logic and remember the programming syntax to write the complete executable program. Thus the six SOLO levels in writing will determine how much students are able to complete in the writing tasks.

The following sections present the specification of individual components, description of SOLO levels and demonstration of how students' responses were evaluated.

a) **Algorithm design:** Algorithm design question 2 in the PST was to ‘Find the smallest number among the three numbers entered by the user and display the result’. A suitable algorithm solution may look like this:

```
Step 1: Start
Step 2: Declare variables a, b and c.
Step 3: Read a, b and c.
Step 4: If a < b
    If a < c
        Display a is the smallest number.
    Else
        Display c is the smallest number.
Else
    If b < c
        Display b is the smallest number.
    Else
        Display c is the smallest number.
Step 5: Stop.
```

The following components were identified in students' response to algorithm questions:

- a) ability to declare variables (E–U)
- b) ability to externally supply inputs (E–S)
- c) ability to show the computation (E–S for Q1) and (H–S for Q2)
- d) ability to show what output is produced (E–S)
- e) ability to write clear and unambiguous instructions (E–U)

A clear and unambiguous instruction will be to use the words *declare*, *read/get*, *display/print* in the instructions.

- f) ability to terminate the algorithm in a finite number of steps (E–U)
- g) ability to write an algorithm in correct logical order (H–U).

A simple example is: Step 3: Read a, b and c instruction should come only after Step 2: Declare variables a, b and c. Overall, the algorithm should complete the individual components listed in this order from a–f.

These individual components are shown in Figure 3.1.

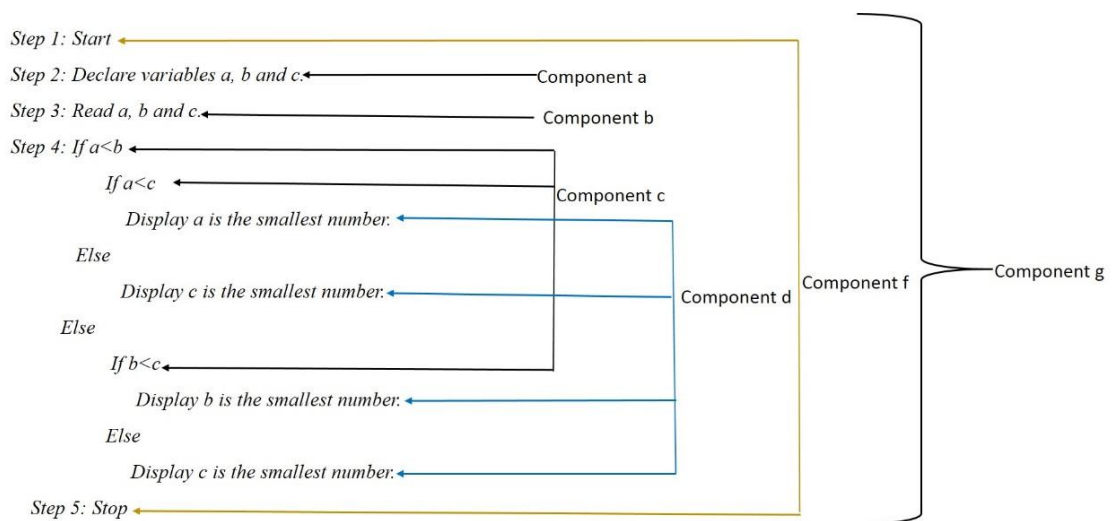


Figure 3.1. Individual components of an algorithm.

The suitable flowchart solution to the algorithm design Question 2 may look like in Figure 3.2.

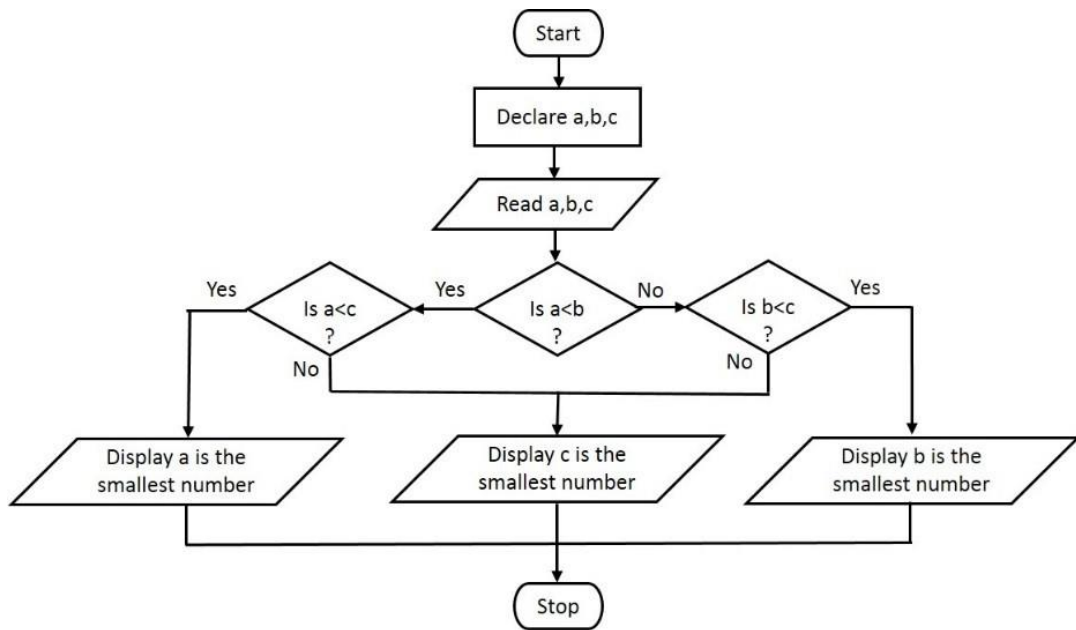


Figure 3.2. Suitable flowchart solution for algorithm design question 2.

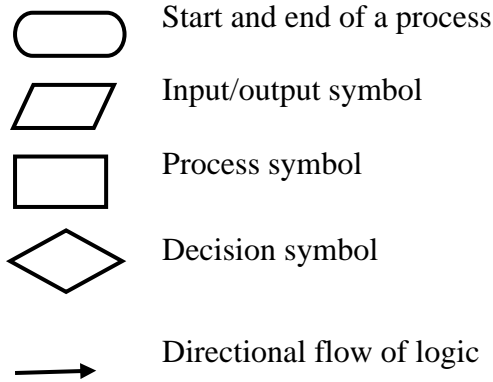
Ten components were identified in students' responses to flowchart questions:

1. ability to declare variables (E–U)
2. ability to externally supply inputs (E–S)
3. ability to show the computation (E–S for Q1) and (H–S for Q2)
4. ability to show what output is produced (E–S)
5. ability to write clear and unambiguous instructions (E–U)
6. as mentioned earlier, a clear and unambiguous instruction will be to use the words *declare*, *read/get*, *display/print* in the instructions
7. ability to terminate the flowchart in a finite number of steps (E–U)
8. ability to draw a flowchart in correct logical order (H–U)



9. ability to draw a flowchart with correct symbols (H–U)

Students were expected to draw flowchart with the correct symbols as shown below:



These individual components are shown in Figure 3.3.

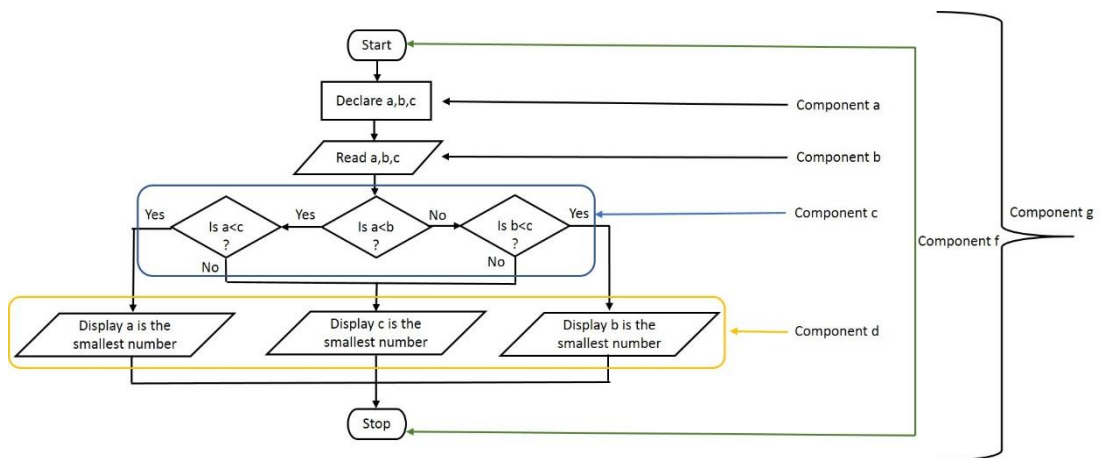


Figure 3.3. Individual components of flowchart.

After identifying the components for algorithm design (algorithm and flowchart) questions, the SOLO classification for algorithm design questions were established, as shown in Table 3.4.

Table 3.4

*SOLO Levels for Algorithm Design Questions*

SOLO Level	Indicator	Raw Score
Relational [R]	Able to complete all components to form a coherent whole	5
Relational error [RE]	Able to complete all components but has some minor errors or omissions	4
Multistructural [M]	Able to complete most components. All H-S and E-S components complete and valid	3
Multistructural error [ME]	Able to complete most components but has some minor errors or omissions	2
Unistructural [U]	Able to complete some components only	1
Prestructural [P]	There are pieces that make no sense or the answer is totally wrong	0
No attempt [N]	The answer is blank	999

Figure 3.4 shows the R (relational) and RE (relational error) SOLO levels and their equivalent scores for the response to the algorithm design Question 2 of the PST. The student response was classified as R, as the response satisfied all the individual components listed above for that question. The student response was classified as RE, as the student omitted the instruction to declare the variables. Students' ability to declare the variables was assigned as E and U; this is the only minor omission as shown in Figure 3.4.

1. START  
 2. Declare variable x, y and z  
 3. Get the values from user and store it in x, y, z.  
 4. If (z < y),  
   If (z < x),  
     display x as smallest.  
   Else,  
     display z as smallest.  
 Else,  
   If (y < z),  
     display y as smallest.  
   Else,  
     display z as smallest.  
 5. STOP

R-5

Start.  
 Enter three numbers a, b and c.  
 Check if (a < b)  
 {  
   if (a < c)  
   {  
     print a is smallest  
   }  
   else  
   {  
     print c is smallest.  
   }  
 }  
 else  
 {  
   if (b < c)  
   {  
     print b is smallest  
   }  
   else  
   {  
     print c is smallest.  
   }  
 }  
 stop.

a) omitted (E-U)  
 b) ✓ (E-S)  
 c) ✓ (H-S)  
 d) ✓ (E-S)  
 e) ✓ (E-U)  
 f) ✓ (E-U)  
 g) ✓ (H-U)

RE-4

Figure 3.4. R and RE SOLO levels and scores for algorithm design question 2 of the PST.

Figure 3.5 shows the M (multistructural) and ME (multistructural error) SOLO levels and their equivalent scores for the response to the algorithm design Question 2 of the PST. The student response was classified as M, as it satisfied most of the individual components, with all of the H-S and E-S components complete and valid. The student response was classified as ME, as it satisfied most of the individual components, with all H-S and E-S components complete but with errors.

a) START  
 2) Enter the value of a, b, and c  
 3) If A < B and A < C  
   Display A  
 Else  
 If B < A and B < C  
   display B.  
 else  
   display c.

a - omitted (E-U)  
 b - ✓ (E-S)  
 c - ✓ (H-S)  
 d - ✓ (E-S)  
 e - ✓ (E-U)  
 f - X (E-U)  
 g = minor error (H-U)

M-31

1) Start  
 2) Get the value of a, b, and c  
 z = a < b? a: b  
 y = z < c? z: b  
 3) Display y.

a) omitted (E-U)  
 b) ✓ (E-S)  
 c) error (H-S)  
 d) ✓ (E-S)  
 e) ✓ (E-U)  
 f) X (E-U)  
 g) ✓ (H-U)

ME-2

Figure 3.5. M and ME SOLO levels and scores for algorithm design question 2 of the PST.

Figure 3.6 shows the U (unistructural) and P (prestructural) SOLO levels and their equivalent scores for the response to the algorithm design Question 2 of the PST. The student response was classified as U, as it satisfied some of the individual components only. The response was classified as P, as it did not satisfy any of the

individual components. Moreover, the response is totally wrong. The P response shows that the student did not understand the question at all.

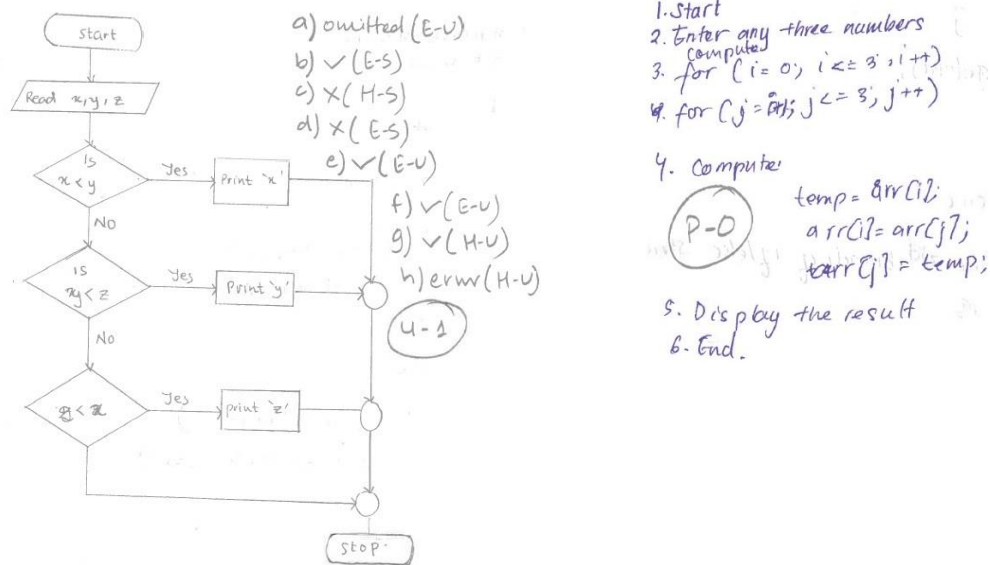


Figure 3.6. U and P SOLO levels and scores for algorithm design question 2 of the PST.

b) **Translating:** Translating question 1 in the PST was to translate the following algorithm into C programming codes.

```

Start
Declare integer variables a, b, BIG,
SMALL.
Read a and b
If a is less than b, then
    BIG = b
    SMALL = a
Else
    BIG = a
    SMALL = b
Write BIG, SMALL
End
    
```

A suitable translated solution may appear as that shown in Figure 3.7.

```
int main()
{
    int a,b, BIG, SMALL;

    scanf("%d %d",&a,&b);

    if(a<b)
    {
        BIG = b;
        SMALL = a;
    }
    else {
        BIG = a;
        SMALL =b;
    }

    printf("BIG =%d, SMALL = %d",BIG, SMALL);
}
```

Figure 3.7. Suitable solution for translating question 1 of the PST.

Six components were identified in students' response to translating question 1:

1. ability to declare variables of correct data types (E-S)
2. ability to read variables from the console (E-S)
3. ability to use if/else statement with correct computation (E-S)
4. ability to show the desired output (E-S)
5. ability to translate into well-structured program in clear logical order (E-U)

Student was expected to translate into well-structured C programming language in a clear logical order, as given in the algorithm design. The correct syntaxes are preferred but not mandatory. However, the translated program should be closer to executable program.

These individual components are shown in Figure 3.8.

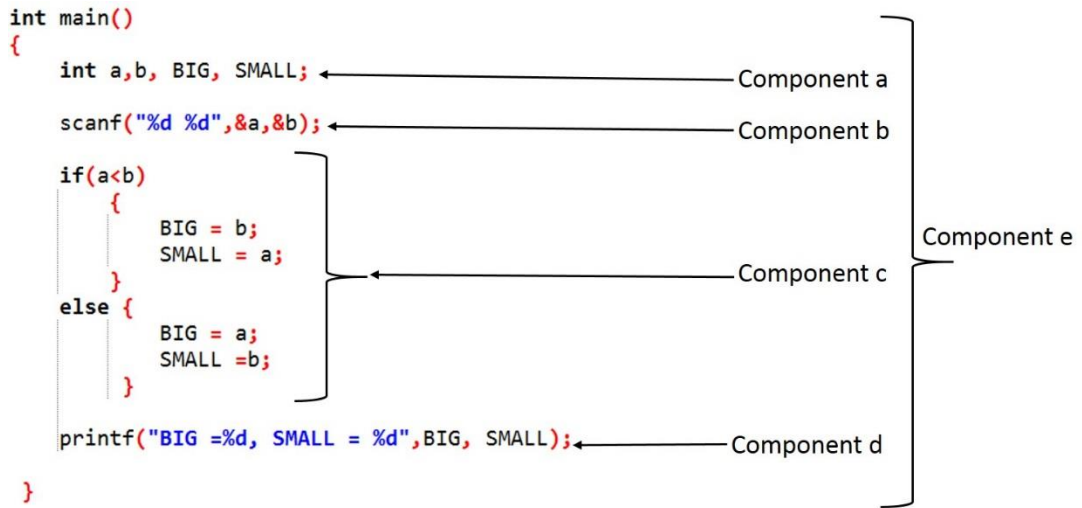


Figure 3.8. Individual components of translating question 1 of the PST.

The SOLO levels shown in Table 3.5, based on the components, were identified on a students’ response to translating question 1 of the PST.

Table 3.5

*SOLO Levels for Translating Questions*

SOLO level	Indicator	Raw Score
Relational [R]	Able to complete all components to form a coherent whole	3
Multistructural [M]	Able to complete most components. All H–S and E–S components complete and valid	2
Unistructural [U]	Able to complete some components only	1
Prestructural [P]	There are pieces that make no sense or the answer is totally wrong	0
No attempt [N]	The answer is blank	999

Student responses to the translating question 1 of the PST were categorised, as shown in Figures 3.9 and 3.10, by examining the individual components and using SOLO descriptions (see Table 3.5).

Figure 3.9 shows the student responses that were assigned R and M SOLO levels. The student response was assigned R, as it fulfils all individual components

listed above and the response was assigned M, as the student was able to complete most of the components, with H-S and E-S components complete. In Figure 3.9, the student has written extra *print* statements at the end of the program, which is not necessary, as the *print* statements to print the desired outputs are written within the *if-else* statement. Thus, this kind of response was classified as M because the student was not able to translate well-structured programs in clear logical order.

```

#include <stdio.h>
main()
{
    int a, b, BIG, SMALL;
    printf("Enter value for a and b:");
    scanf("%d %d", &a, &b);
    if (a < b)
    {
        BIG = b;
        SMALL = a;
    }
    else
    {
        BIG = a;
        SMALL = b;
    }
    printf("%d %d", BIG, SMALL);
}

```

a) ✓(E-S)  
b) ✓(E-S)  
c) ✓(E-S)  
d) ✓(E-S)  
e) ✓(E-U)

R-3

```

#include <stdio.h>
main()
{
    int a, b;
    printf("enter the value of a, b:");
    scanf("%d %d", &a, &b);
    if (a < b)
    {
        printf("BIG = %d", b);
        printf("SMALL = %d", a);
    }
    else
    {
        printf("BIG = %d", a);
        printf("SMALL = %d", b);
    }
    printf("BIG, SMALL.");
}

```

a) ✓(E-S)  
b) ✓(E-S)  
c) ✓(E-S)  
d) ✓(E-S)  
e) ✗(E-U)

M-2

Figure 3.9. R and M SOLO levels and scores for translating question 1 of the PST.

Figure 3.10 shows the student responses that were assigned U and P SOLO levels. The student response was assigned U, as the response fulfilled some individual components only and the response was assigned P, as the student was not able to translate the program correctly. For instance, the algorithm stated the variables *a* and *b* to be of integer datatype (*int a, b;*) but the student declared them as character datatypes (*char a, b;*), which shows that the student has not understood the algorithm clearly.

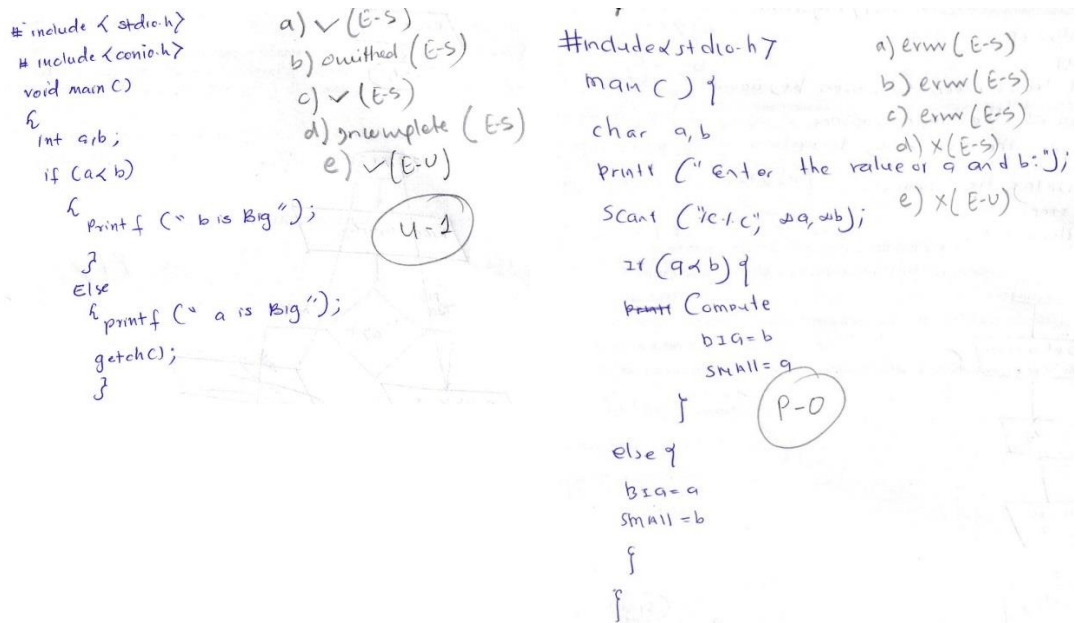


Figure 3.10. U and P SOLO levels and scores for translating question 1 of the PST.

c) **Tracing:** Tracing question 2 in the PST was to manually trace the following piece of code and answer, parts a) and b).

```

int i, j, x = 0;

for (i = 0; i < 3; ++i)
{
    for (j = 0; j < i; ++j)
    {
        x += (i + j - 1);
    }
}
printf("x = %d", x); ← Line 12

```

Write the value of x when i = 1

Write the value of x at line 12.

The answers are: a) 0 and b) 3.

The SOLO levels for tracing questions are shown in Table 3.6.



Table 3.6

*SOLO Levels for Tracing Questions*

SOLO level	Indicator	Raw Score
Relational [R]	Able to provide correct solution to parts a) and b)	3
Multistructural [M]	Able to provide solution to parts a) and b) with minor errors or omissions	2
Unistructural [U]	Only one part of the two completed correctly	1
Prestructural [P]	There are pieces that makes no sense or the answer is totally wrong	0
No attempt [N]	The answer is blank	999

Figure 3.11 shows how student responses to tracing question 2 of the PST were categorised based on SOLO descriptions listed in Table 3.6.

The response in Figure 3.11 was assigned R, as the student was able to show correct output to parts a and b, while the response was assigned M, as the student was able to show the correct output only for part a; the output in part b was incomplete. The value of  $x$  is updated from 0, 1 and 3 and the final answer printed is 3.

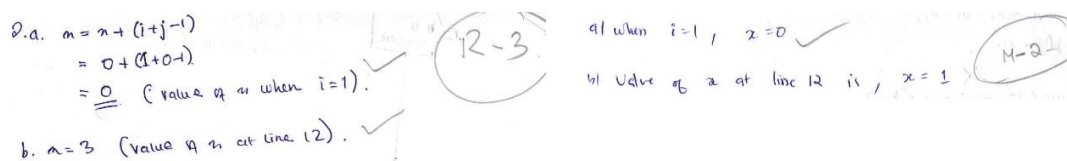


Figure 3.11. R and M SOLO levels and scores for tracing question 2 of the PST.

The response in Figure 3.12 was assigned U, as the student was able to show correct output for only one part of the question. The response was assigned P, as the student was not able to show the correct output for both the parts.

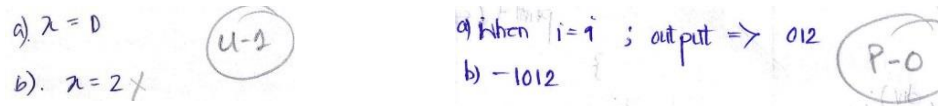


Figure 3.12. U and P SOLO levels and scores for tracing question 2 of the PST.

**d) Explaining:** Explaining question 2 of the PST was to explain in plain English the purpose of the following piece of code:

```

#include<stdio.h>
int main()
{
    int i,c=0;

    int a[10]= {2,3,4,5,7,8,9,12,34,6};

    for(i=0;i<10;i++)
    {
        if(a[i]%2==0)
        {
            c++;
        }
    }

    printf("%d",c);
    return 0;
}

```

The suitable solution may look like this:

*The code counts the number of even numbers in a given array.*

Table 3.7 shows the SOLO levels for the explaining questions.

Table 3.7

*SOLO Levels for Explaining Questions*

SOLO level	Indicator	Raw Score
Relational [R]	Able to provide a summary of what the code does in terms of its purpose	5
Relational error [RE]	Able to provide a summary of what the code does in terms of its purpose but with some minor errors or omissions	4
Multistructural [M]	Able to provide a line-by-line description of all the code	3
Multistructural error [ME]	Able to provide a line-by-line description of most of the code but with some minor errors or omissions	2
Unistructural [U]	Able to provide description of one portion of the code	1
Prestructural [P]	There are pieces that make no sense or the answer is totally wrong	0
No attempt [N]	The answer is blank	999

Figure 3.13 shows how student responses to the explaining question 2 of the PST were categorised as R and RE based on the SOLO descriptions listed in Table 3.7. The response in Figure 3.13 was assigned R, as the student was able to provide a summary of what the given piece of code did, while the response was assigned RE, as the student was able to provide an R response but also provided extra information not given in the code. For example, ‘The purpose of the code is to print the even elements in the array’ was not given in the code. The second half of the response, ‘to count how many even elements there are’, is an R response. However, as the student was unable to provide a summary of the purpose of the code, it was assigned RE.

The program is design for finding <sup>total</sup> number of 'even' number in a given array. (R-5)

The purpose of this code is to print the even elements present in the array and to count how many even elements there are. (RE-4)

Figure 3.13. R and RE SOLO levels and scores for explaining question 2 of the PST.

Figure 3.14 shows how the student responses to the explaining question 2 of the PST were categorised as M and ME. The student response was assigned M, as the student provided a line-by-line description of all the code instead of a summary. The response was assigned ME, as the student was able to provide a line-by-line description of most of the code, but with some minor errors or omissions.

Variable i and c is declared where i's value is 0. The Array a[10] is declared with the values of 2, 3, 4, 5, 7, 8, 9, 12, 34, and 6. Then a loop is declared for 10 times and from to see if the value of a[i] array has any even number, if so c++ is processed and finally value of c is printed after the end of loop. (M-3)

Variables i, c and a is declared and initialized. for loop is used to read numbers from 0 to 9. If statement is used to check the even numbers between 0 and 9. printf() function is used to display even numbers between 0 and 9. (ME-2)

Figure 3.14. M and ME SOLO levels and scores for explaining question 2 of the PST.

Figure 3.15 shows how student responses to the explaining question 2 of the PST were categorised as U and P. The student response was assigned U, as the student was able to provide a description of only one portion of the code. The response was assigned P, as the student provided pieces of information that were not correct.

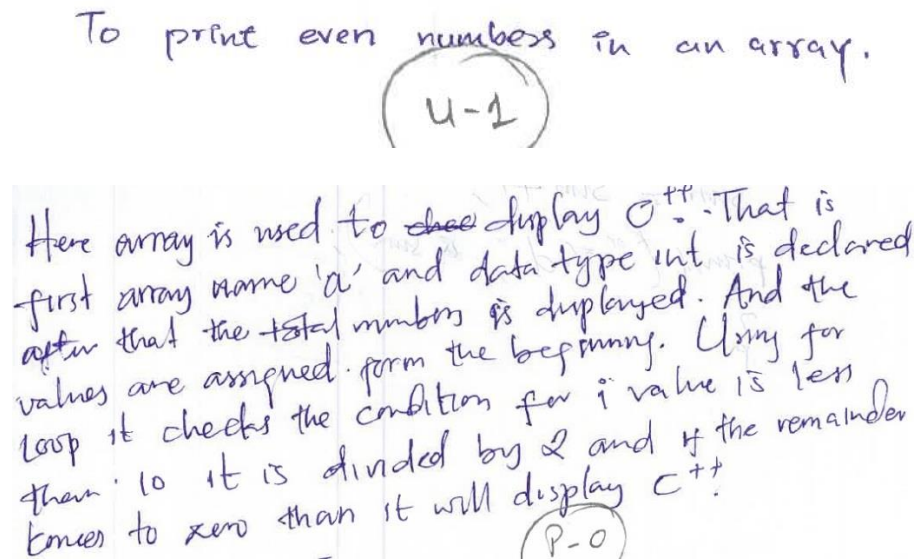


Figure 3.15. U and P SOLO levels and scores for explaining Question 2 in a PST.

e) **Writing:** Writing question 2 in the PST was to 'Write a program in C that calculates the sum of every third integer, beginning with  $i = 2$ ' (i.e., calculate the sum of  $2 + 5 + 8 + 11 + \dots$ ) for all values of  $i$  that are less than 30. A suitable solution may be:

```
#include <stdio.h>

int main()
{
    int i, sum;

    sum=0;

    for(i=2; i<30; i=i+3)
    {
        sum+=i;
    }
    printf("The sum = %d", sum);

    return 0;
}
```

Seven components were identified to students' response to writing question 2:

1. ability to declare variables of correct data types (E–S)
2. ability to initialise a variable used to accumulate *sum* in the program (H–U)
3. ability to formulate correct loops (H–U)

Use of any loops such as *while* or *do-while* and not necessary the *for* loop shown here

4. ability to compute the *sum* correctly (H–S)
5. ability to print the *sum* correctly (E–U)
6. ability to write well-structured program in clear logical order (H–U).

The student was expected to write codes using C programming language in a well-structured manner with clear logical order closer to the suitable solution shown previously.

Figure 3.16 shows these individual components.

```
#include <stdio.h>

int main()
{
    int i, sum; ← Component a
    sum=0; ← Component b
    for(i=2; i<30; i=i+3) ← Component c
    {
        sum+=i; ← Component d
    }
    printf("The sum = %d", sum); ← Component e
    return 0;
}
```

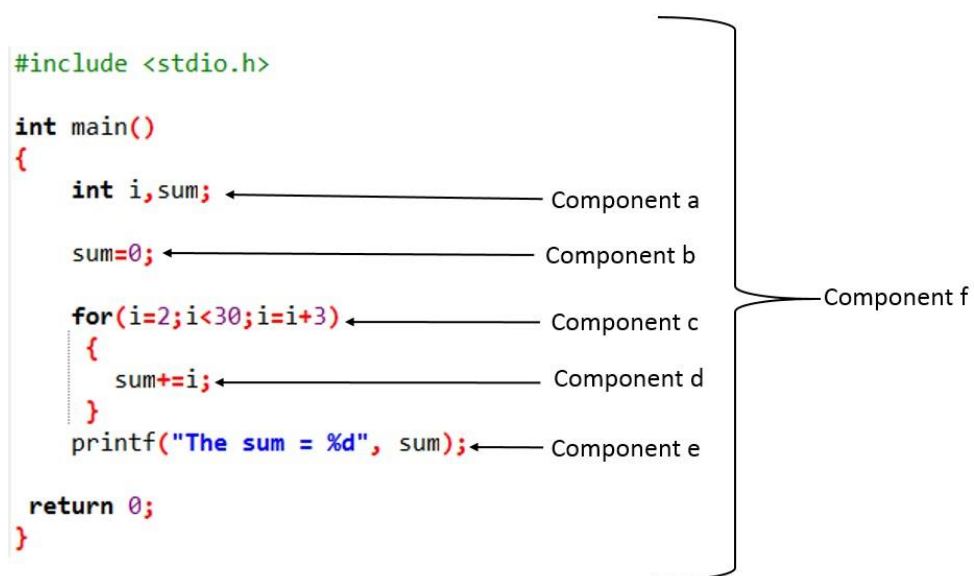


Figure 3.16. Individual components of writing question 2 of the PST.

Thus, the SOLO description for the writing questions in the PST was established as shown in Table 3.8, based on the individual components shown previously.

Table 3.8

*SOLO Classification for Writing Questions*

SOLO Level	Indicator	Raw Score
Relational [R]	Able to complete all components as a coherent whole	6
Relational Error [RE]	Able to complete all components but has some minor syntax or logic errors or omissions	5
Multistructural [M]	Able to complete most components. All H–S and E–S components complete and valid	4
Multistructural error [ME]	Able to complete most components but with some minor syntax or logic errors or omissions	3
Unistructural [U]	Able to complete some components only	2
Unistructural error [UE]	Able to complete some components only with some syntax or logic errors	1
Prestructural [P]	There are pieces that make no sense or the answer is totally wrong	0
No attempt [N]	The answer is blank	999

Figure 3.7 shows the relational SOLO levels and their equivalent score for the R and RE responses to writing question 2 of the PST. The student response was classified as R, as the response satisfies all individual components listed. Although the second response completed all components, there was a missing statement to *print* the sum. Thus, the response was classified as RE.

```

#include <stdio.h>
void main()
{
  int i, sum=0;
  for(i=2; i<30; i=i+3)
  {
    sum = sum+i;
  }
  printf("%d", sum);
  getch();
}

```

a) ✓ (E-S)  
 b) ✓ (H-U)  
 c) ✓ (H-U)  
 d) ✓ (H-S)  
 e) Omitted (E-U)  
 f) ✓ (H-U)

RE-5

R-6

Figure 3.17. R and RE SOLO levels and scores for writing question 2 in the PST.

Figure 3.18 shows the M and ME SOLO levels and their equivalent score for the response to the writing question 2 of the PST. The response was classified as M, as it satisfied all components of H-S and E-S but with a minor omission or error, with U component. Accordingly, if the response satisfied all components of H-S and E-S but with error or omissions, the response was classified as ME.

```

#include <stdio.h>
main() {
  int i, sum=0;
  for (i=2; i<30; i+i+3) {
    sum = sum+i;
  }
  printf("The sum is %d", sum);
}

```

a) ✓ (E-S)  
 b) ✓ (H-U)  
 c) error (H-U)  
 d) ✓ (H-S)  
 e) ✓ (E-U)  
 f) ✓ (H-U)

M-4

```

#include <stdio.h>
main()
{
  int i, sum=0;
  for (i=2; i<30; i++) {
    sum = sum+i;
  }
  i=i+3;
  printf("%d", sum);
}

```

a) ✓ (E-S)  
 b) ✓ (H-U)  
 c) error (H-U)  
 d) ✓ (H-S)  
 e) ✓ (E-U)  
 f) ✗ (H-U)

ME-3

Figure 3.18. M and ME SOLO levels and scores for writing question 2 of the PST.



Figure 3.19 shows the U and UE (unistructural error) SOLO levels and their equivalent scores for the response to writing question 2 of the PST. The sample student response was classified as U, as the student was able to complete only some of the components. Similarly, the second sample response completed only some of the components, with syntax and logic errors. Such responses were classified as UE.

<pre>#include &lt;stdio.h&gt; main() {   int i, n=30, sum=0;   for(i=2; i&lt;30; i+=2);   {     sum = sum + (i+3);   }   printf("Sum of every third integer = %d", sum); }</pre>	<p>a) ✓ (E-S)          b) ✓ (H-U)          c) ✓ (H-U)          d) ✗ (H-S)          e) ✓ (E-U)          f) ✓ (H-U)</p> <p style="text-align: center;">(U-0)</p>	<pre>#include &lt;stdio.h&gt; main() {   int i, n=30, sum;   for(i=2; i&lt;30; i+=2)   {     sum = i + 2;   }   printf("Sum ", sum); }</pre>	<p>a) ✓ (E-S)          b) ✗ (H-U)          c) ✓ (H-U)          d) ✗ (H-S)          e) ✓ (E-U)          f) ✓ (H-U)</p> <p style="text-align: center;">(UE-1)</p>
--	--	--	---

Figure 3.19. U and UE SOLO levels and scores for writing Question 2 of the PST.

Figure 3.20 shows the P SOLO levels and their equivalent score for the response to writing question 2 in the PST. As shown, the student provided a completely incorrect answer to the questions. Instead of writing a program, the student drew a flowchart, which shows the student did not understand the question clearly. Such types of responses were classified as P.

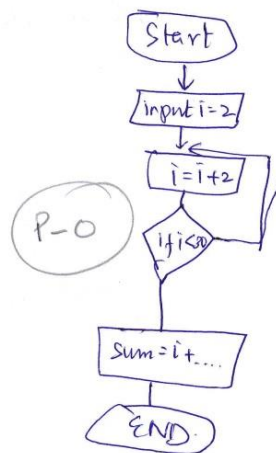


Figure 3.20. P SOLO levels and scores for writing Question 2 in the PST.

The assessment of PST answer scripts using SOLO taxonomy (as shown in this chapter) for the remaining questions under algorithm design, translating, tracing, explaining and writing is shown in Appendix C.

### **3.8 Ethical Considerations**

Several ethical issues were considered when gathering data for this study. Following the guidelines of informed consent (Cohen, 2017), the information letter (see Appendix E) that explained the purpose of the research and procedures along with the consent forms (Appendix D) was given to participants before any data collection activities were carried out.

The first ethical consideration considered voluntarism in participation in the PST and survey. Voluntarism ensures that participants have the right to decide whether to take part in the PST/survey. Volunteer student participants were then provided with the information letter, which consisted of a brief summary of the study and how their participation would assist in achieving the purpose of this research. It also contained information in regard to the privacy and confidentiality of students' details and assurances that information would remain confidential and names would not be identified or produced in any documents.

The second ethical consideration was in regard to the anonymity of student participants. Complete anonymity was not possible within the context of this study. Thus, student participants were asked to write their student registration number and enrolment program only for the PST and survey. Participants were also verbally informed that their student registration numbers would not be reported in this thesis or in any other papers. Instead, a unique identifier such as S001, S002 or S003 would be assigned if needed. The student registration number was required for analysing data

across various data collection methods. Once students volunteered to participate in this study and read the information letter, they were provided with the consent form to sign.

The third ethical consideration involved power and position. As Bhutan is a country of rich culture where there is great respect for elders and people in higher positions, the researcher was concerned that students may have volunteered out of respect for their lecturer, even if they did not want to. If this were to occur, students' might feel pressured and may not have been able to provide correct answers. To avoid this situation, the researcher briefed the volunteer lecturers prior to the administration of the PST to reduce power difference and enable students to have power over decision-making (Cohen, 2017). Students were also assured that their performance in the PST would not affect their grades, and data from the PST would be used solely for the purpose of this study. Similarly, when the researcher visited the three colleges of RUB to administer the survey and interviews, students were informed by their lecturers of their right to participate in the survey or not. Prior to survey, the researcher briefly introduced the study and explained the contents of the information letter and consent form. Students were made to feel important and encouraged to be at ease while filling out the survey. Verbal consent was obtained from students selected for individual and group interviews. Where selected students refused to participate, other students were invited to participate.

The fourth issue was withdrawal from participation. This ensured that participants could withdraw and discontinue their participation in the study at any stage without prejudice to them. This was stated in the information letter.

Finally, lecturer participants in the survey were provided with the information letter and consent form. Verbal consent was obtained from lecturers prior to the

interview. The researcher verbally informed lecturers that the data collected would be used solely for the purpose of research and would not affect their academic career.

Ethical approval (HRE2016-0318) was granted by the Human Research Ethics Committee of Curtin University (see Appendix F).

### **3.9 Data Analysis**

Data were first entered in Microsoft Excel and imported into SPSS, a statistical data analysis package. SPSS was used to analyse the quantitative data collected via the data collection methods described in Section 3.6. The quantitative data were first analysed using univariate statistical analysis for descriptive statistics. Subsequently, the quantitative data were analysed using bivariate correlations analysis and multiple linear regression, which included assumption testing. Path analysis was also conducted. The results of the descriptive statistics, bivariate correlation analysis, multiple linear regression and path analysis are presented and discussed in Chapters 4, 5 and 6.

The qualitative data were first transcribed into Microsoft Excel and manually analysed by focusing on the areas related to the research questions. For a particular research question, the researcher was required to read all student responses and summarise them. New ideas/areas that emerged that were not related to the research questions were also noted and summarised. The results of the student and lecturer qualitative data are presented in Chapter 4 and 5.

A second assessor was invited to evaluate sample students' PST responses to obtain inter-rater reliability (IRR) to check the consistency of the PST marking criteria with the researcher's marking. IRR was evaluated using intra-class correlation coefficient. Results are presented in Chapter 5 (see Section 5.4).

### **3.10 Summary**

This chapter has provided a detailed description of the research methodology used in this study and has explained the rationale behind using a mixed methods in collecting both quantitative and qualitative data. The research questions that guided this study were presented. Details about the data collection methods to address each research question was shown and ethical considerations made prior to data collection were described. The data analysis process was also summarised. Chapters 4, 5 and 6 present the analysis results of the quantitative and qualitative data.

## **Chapter 4: Students' Quantitative Univariate and Qualitative**

### **Results**

#### **4.1 Introduction**

Chapter 3 described the preparation for data collection and outlined the research questions. The instruments were discussed and presented. This chapter presents the student quantitative univariate programming skills test (PST) and survey results, followed by student qualitative results.

The first section presents the measure of student success in Introduction to Computer Programming (CS1) that was used in this study. Student performance (student success in CS1) was measured by performance in the PST, final semester examination (FSE) and overall semester (OS) performance. Students' quantitative univariate results from the PST and survey data are presented next. As mentioned previously, the PST consisted of questions that interrogate students' ability across five programming skills: algorithm design, translating, tracing, explaining and writing. The descriptive statistics for those programming skills are presented here. This is followed by the student survey and interview results for the areas covered in the students' questionnaire, such as prior computing experience; first programming language, programming paradigm, programming environment, and teaching/learning methods and practices that have improved learning in CS1; Year 12 (Y12) performance in mathematics, physics and chemistry; order of programming skills to be learnt in CS1; order of contribution of programming skills to student performance as perceived and experienced by students in CS1; approaches to learning in CS1 and suggestions for the improvement of teaching/learning of CS1 at RUB. The independent sample *t*-test was also conducted to validate the results.

## **4.2 Measure of Student Performance in CS1**

This section presents the data used to measure student performance in CS1. In this study, student performance was the dependent variable. Student performance in CS1 was measured by their performance in the PST, FSE and the OS.

### **Students' performance in the PST, FSE and OS**

Students' PST results were collected from the PST administered to all student participants from Sherubtse College (SC), Jigme Namgyel Engineering College (JNEC) and College of Science and Technology (CST), who were enrolled in the science and engineering program and who had completed CS1 in their first semester (July–November 2016).

Students' FSE scores were collected from the respective lecturers who taught CS1 in July–November 2016 in the three colleges at Royal University of Bhutan (RUB). FSEs were independently administered by an individual college. Similarly, OS scores were collected from the respective lecturers and the student survey. Table 4.1 shows the descriptive statistics of the PST, FSE and OS. There is little difference between the means of PST (52.44) and FSE (55.77) compared to the mean of OS (61.95). The standard deviations of PST (17.40) and FSE (15.54) were fairly similar compared to the standard deviation of OS (11.63). This could be because the PST and FSE were written in an examination settings while the OS is the total of continuous assessment (CA) (50 percent) and FSE (50 percent). CA consists of mid-semester examination, class tests, assignments and practical examinations.

Table 4.1

*Descriptive Statistics of Student Performance (PST, FSE, OS)*

Student Performance	n	Mean	Standard Deviation
PST	292	52.44	17.40
FSE	285	55.77	15.54
OS	270	61.95	11.63

Note: n is not equal, as missing data were omitted.

### 4.3 Student Quantitative Univariate Results

This section presents the quantitative univariate results of student PST and surveys. A total of 327 students participated in this study out of 342 students. The response rate was 96 percent which is close to 100 percent. The reason behind this high response rate has been discussed in Section 3.8 under ethical consideration. As the Bhutanese socio-cultural environment is unique in terms of respect for elders, people in higher positions and lecturers, the students may have participated out of respect to their lecturer. However, the researcher made every possible way to minimise this power difference and enable students to have autonomy over their decision. As a result of this a small number of students did choose not to participate.

Of 327 students, 292 students participated in the PST and 277 participated in the survey. The number of students who participated in both the PST and survey was 238.

Table 4.2 shows the number of students who participated from seven programmes in three colleges at RUB. As discussed in Chapter 3, SC offers B.Sc. in Computer Science—the number of the students who participated was 27. JNEC offers a Diploma in Computer Hardware and Networking—the number of students who



participated was 43. Finally, CST offers five engineering programs—the number of students who participated was 257, the highest number of students from any of the three RUB colleges.

Table 4.2

*Student Representation in this Study from Three RUB Colleges*

College	Program	Male	Female	Total
CST	B.Eng. in Electronics and Communication	24	6	30
	B.Eng. in Electrical	47	18	65
	Bachelor of Architecture	11	7	18
	B.Eng. in Information Technology	26	12	38
	B.Eng. in Civil	70	36	106
JNEC	Diploma in Computer Hardware and Networking	22	21	43
SC	B.Sc. Computer Science	20	7	27
Total students		220	107	327

### 4.3.1 PST results

This section presents the summary statistics of data collected via the PST. The raw scores of the equivalent SOLO levels were used in the analysis.

A total of 292 students in three colleges at RUB undertook the PST specifically devised for this study. Females represented about 34 percent ( $n = 99$ ) of the population and males about 66 percent ( $n = 193$ ). CST students represented 76 percent ( $n = 223$ ), JNEC 15 percent ( $n = 43$ ) and SC nine percent ( $n = 26$ ) of the sample.

As discussed in Chapter 3, PST data were collected using a printed PST paper administered by lecturers who volunteered to assist the researcher. The PST was administered after student participants had completed one semester of instruction in CS1 and before the FSE. The PST was devised by the researcher in consultation with lecturers to interrogate students' abilities across five programming skills: algorithm

design, translating, tracing, explaining and writing. The PST consisted of 10 questions. There were two questions for each programming skill. Subsequently, the SOLO taxonomy was adapted to evaluate students' responses to PST questions (as discussed and presented in Chapter 3). Student scores were converted to a percentage for ease of comparison.

Table 4.3 presents the college-wise summary statistics of the students' performance in the PST. On average, CST students performed best in the PST (54.23), followed by SC (45.16) and JNEC (45.16). The reason for the higher mean at CST could be that most top performing students' first option is to obtain entry into one of the engineering degree programs in CST, then a computer science degree program in SC and finally, the diploma program at JNEC.

Table 4.3

*College Summary Statistics in PST*

College	n	Mean	Standard Deviation
CST	223	54.23	18.19
SC	26	49.18	13.11
JNEC	43	45.16	13.54

Table 4.4 presents the summary statistics for each college with participant representatives in the PST. On average, students enrolled in the architecture program performed well in the PST (63.34) compared to students enrolled in other programs. This may be because only the top performing students opted for this program, so they may have done well in CS1. Students who were qualified but were not selected based on merits in the engineering program opted for Computer Science or a Diploma in Computer Hardware and Networking—and their PST average was 49.18 and 45.16 respectively.

Table 4.4

*Programme Summary Statistics in PST*

Program	n	Mean	Standard Deviation
Bachelor of Architecture	11	63.34	17.99
B.Eng. in Electronics and Communication	27	59.72	13.86
B.Eng. in Information Technology	33	55.06	14.42
B.Eng. in Civil	98	53.40	20.84
B.Eng. in Electrical	54	50.62	15.86
B.Sc. Computer Science	26	49.18	13.11
Diploma in Computer Hardware and Networking	43	45.16	13.54

Table 4.5 shows the summary statistics of the overall PST scores in each of the programming skills. On average, students performed well in algorithm design, translating, explaining and tracing, followed by writing. Since algorithm design is a skill that students learnt before other programming skills, and writing is a skill that student should be able to attain at the end of the course, the results reported here were consistent with the pattern that would be expected. That is, algorithm design has the highest mean, followed by translating, explaining, tracing and writing.

Table 4.5

*Summary Statistics of the Programming Skills and Overall PST Score*

Programming Skills	n	Mean	Standard Deviation
Algorithm design	290	61.72	23.78
Translating	291	59.51	28.08
Explaining	287	56.48	32.70
Tracing	288	46.76	23.76
Writing	278	41.67	21.62
Test score	292	52.44	17.40

Note: n is not equal across categories, as missing data were omitted. Some participants did not respond.

### 4.3.2 Quantitative survey results

This section presents the summary of data collected via survey questionnaire. The survey data presented here represent the responses of 277 students in three colleges of RUB.

Figure 4.1 shows the number of students who participated in the survey across seven programs. CST represented 77 percent ( $n = 213$ ), JNEC 13 percent ( $n = 37$ ) and SC 10 percent ( $n = 27$ ) respectively. Females represented about 34 percent ( $n = 94$ ) and males about 66 percent ( $n = 183$ ).

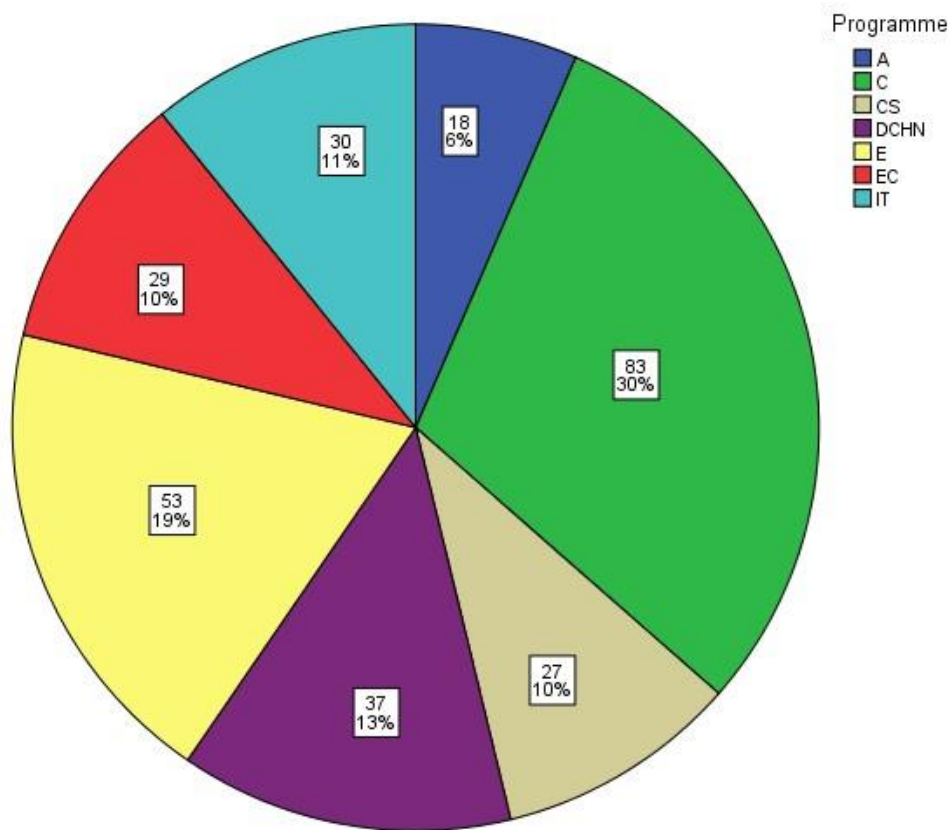


Figure 4.1. Number of student participants from seven programs in the survey.

Data were collected using a printed survey questionnaire, which was administered by the researcher to student participants. The areas covered in the questionnaire were: prior computing experience; Y12 performance in mathematics, physics and chemistry; programming environments used in CS1, first programming

language to be taught in CS1, first programming paradigm and teaching/learning methods and practices that have improved students' performance in CS1; order of programming skills to be learnt in CS1; contribution of programming skills to success in CS1; approaches to learning in CS1; and suggestions for improvements in teaching and learning in CS1. The question on suggestions to improve teaching and learning was open ended, so the results will be covered in Section 4.4.1.

The survey questionnaire consisted of 15 questions. Questions were closed-ended, single and multiple responses, scaled responses and open-ended questions. Sections 4.3.2.1–4.3.2.7 report on the summary statistics of the areas covered in the survey questionnaire.

#### ***4.3.2.1 Prior computing experience***

Prior computing experience was categorised as prior programming and prior non-programming computer experience. Question 1 asked students to indicate whether they had prior computer programming experience. They may have gained prior programming experience as a result of taking any programming courses in high school or private training courses prior to learning CS1. Of the 277 student participants, only 39 had some computer programming experience. In Question 2, these 39 students were asked to identify which computer programming language/s that they had experienced. Students' responses indicated some experience in C ( $n = 24$ ), C++ ( $n = 11$ ), Java ( $n = 7$ ), JavaScript ( $n = 3$ ) and Python ( $n = 1$ ). Students were able to select more than one language in which they had experience.

Table 4.6 shows the results of a  $t$ -test and descriptive statistics of FSE, OS and PST by prior programming experience. The only significant difference ( $t$ -test) between means for the groups was found when comparing groups on their OS score. Students with prior programming experience ( $n = 33$ ) scored OS  $M = 65.72$  ( $SD = 9.70$ ). By

comparison, students without prior programming experience ( $n = 218$ ) scored a lower mean for OS,  $M = 61.24$  ( $SD = 11.80$ ).

The relevant distributions were sufficiently normal for the purposes of conducting a  $t$ -test (i.e., skew  $< |2.0|$  and kurtosis  $< |9.0|$ ) (Schmider, Ziegler, Danay, Beyer, & Bühner, 2010). Additionally, the assumption of homogeneity of variances was tested and satisfied via Levene's  $F$  test,  $F(249) = 0.161$ ,  $p = 0.689$  ( $p > 0.05$  indicates unequal variances). The independent sample  $t$ -test was associated with a statistically significant effect,  $t(249) = 2.08$ ,  $p = 0.039$ . Thus, students with prior programming experience were associated with a statistically significantly larger mean in OS than those who had no prior programming experience.

Table 4.6

*Results of t-Test and Descriptive Statistics of FSE, OS and PST by Prior*

*Programming Experience*

Student Performance	Prior Programming Experience						95% CI for		
	Yes			No			Mean		
	M	SD	n	M	SD	N	Difference	t	Df
FSE	58.29	11.98	37	55.98	14.43	220	[-2.63,7.25]	.92	255
OS	65.73	9.70	33	61.24	11.80	218	[.23,8.74]	2.08*	249
PST	57.05	17.48	32	52.44	15.77	208	[-1.38,10.60]	1.52	236

Note: \* $p < 0.05$ ,  $n=327$ . Total student participants in PST and survey. Missing data were omitted.

While there were statistically significant differences in means between the students with and without prior programming experience in OS, no statistical difference existed between students with and without prior programming experience in terms of FSE and PST scores.

Question 3 asked students to indicate their prior non-programming computer experiences by stating their perceived average number of hours per week that they had spent on information searches using the internet, computer games and application

software such as Office, spreadsheets, presentation programs and databases. The purpose of this question was to investigate whether such activities benefited students in learning CS1. There was a difference in means between non-programming computer experiences categorised into two groups based on the average number of hours spent on each activities on the scores of student performance. However, an independent samples *t*-test showed that this was not statistically significant. This suggests that these non-programming computer activities were not important in terms of predicting success in CS1. The correlational analysis in Chapter 6 will further confirm these results.

#### ***4.3.2.2 Y12 performance in mathematics, physics and chemistry***

Question 4 asked students to record their Y12 scores in mathematics, physics and chemistry. The mean of students' Y12 performance in mathematics was 78.8, physics was 73.5 and chemistry was 67.7. Based on the means of students' performances in mathematics, physics and chemistry, students were grouped for each subject into two categories: those who scored more than the mean (*A*) and those who scored less or equal to the mean (*B*). The reason for categorisation was to examine whether Y12 performance in mathematics, physics and chemistry had any impact on student performance in CS1.

Table 4.7 shows the results of the *t*-test and descriptive statistics of FSE, OS and PST by students' Y12 performance in mathematics for categories *A* and *B*. Students with Y12 performance in mathematics for category *A* ( $n = 150$ ) scored as follows for FSE:  $M = 59.16$  ( $SD = 13.32$ ). By comparison, students' Y12 performance in mathematics for category *B* ( $n = 107$ ) scored as follows for FSE:  $M = 52.77$  ( $SD = 14.43$ ). To test the hypothesis that students' Y12 performance in mathematics for categories *A* and *B* were associated with statistically significantly different mean

in FSE, an independent sample  $t$ -test was performed. Distributions for  $A$  and  $B$  were sufficiently normal for the purposes of conducting a  $t$ -test (i.e., skew  $< |2.0|$  and kurtosis  $< |9.0|$ ) (Schmider et al., 2010). Additionally, the assumption of homogeneity of variances was tested and satisfied via Levene's  $F$  test,  $F(255) = 0.206$ ,  $p = 0.651$ . The independent sample  $t$ -test was associated with a statistically significant effect,  $t(255) = 3.664$ ,  $p = 0.000$ . Thus, students who scored highly in Y12 mathematics had a statistically significantly larger mean in FSE than those who scored less.

Table 4.7

*Results of t-Test and Descriptive Statistics of FSE, OS and PST by Students' Y12 Performance in Mathematics for Categories A and B*

Student Performance	Average Y12 Mathematics Score						95% CI for		
	A			B			Mean		
	M	SD	n	M	SD	N	Difference	t	Df
FSE	59.16	13.32	150	52.77	14.43	107	[2.95,9.83]	3.66*	255
OS	64.99	9.89	141	57.81	12.29	110	[4.42,9.94]	5.13*	249
PST	56.47	16.65	135	49.16	14.64	103	[3.25,11.34]	3.54*	236

Note: \* $p < 0.05$ ,  $n=327$ . Total student participants in test and survey. Missing data were omitted.

In Table 4.7, we can observe that students with better scores in Y12 mathematics were associated with a statistically significantly larger mean in OS than those with lower scores in Y12 mathematics. In addition, students with better scores in Y12 mathematics were associated with a statistically significantly larger mean in the PST than with those lower scores in Y12 mathematics.

Table 4.8 shows the results of the  $t$ -test and descriptive statistics of FSE, OS and PST by students' Y12 performance in physics for categories  $A$  and  $B$ . Students with Y12 performance in physics for category  $A$  ( $n = 146$ ) scored as follows for FSE:  $M = 58.17$  ( $SD = 13.49$ ). By comparison, students' Y12 physics score for category  $B$  ( $N = 111$ ) scored as follows for FSE:  $M = 54.29$  ( $SD = 14.67$ ). To test the hypothesis that students' Y12 performance in physics for categories  $A$  and  $B$  were associated with



statistically significantly different mean in FSE, an independent sample *t*-test was performed. Distributions for *A* and *B* were sufficiently normal for the purposes of conducting a *t*-test (i.e., skew < |2.0| and kurtosis < |9.0|) (Schmider et al., 2010). Additionally, the assumption of homogeneity of variances was tested and satisfied via Levene's *F* test,  $F(255) = 0.093$ ,  $p = 0.761$ . The independent sample *t*-test was associated with a statistically significant effect,  $t(255) = 2.197$ ,  $p = 0.029$ . Thus, students who scored highly Y12 physics had a statistically significantly larger mean in FSE than those who scored less.

Table 4.8

*Results of t-Test and Descriptive Statistics of FSE, OS and PST by Students' Y12 Performance in Physics for Categories A and B*

Student Performance	Average Y12 Physics Score						95% CI for		
	A			B			Mean		
	M	SD	n	M	SD	n	Difference	T	Df
FSE	58.18	13.50	146	54.30	14.67	111	[.401,7.35]	2.19*	255
OS	64.21	10.26	139	58.91	12.41	112	[2.48,8.12]	3.70**	249
PST	55.22	16.87	131	50.97	15.07	107	[.12,8.37]	2.02*	236

Note: \* $p < 0.05$ , \*\* $p < 0.05$  n=327. Total student participants in test and survey. Missing data were omitted.

From Table 4.8, we can observe that the students with better scores in Y12 physics were associated with a statistically significantly larger mean in OS than those with lower scores in Y12 physics. Similarly, students with better scores in Y12 physics were associated with a statistically significantly larger mean in PST than those with lower scores in Y12 physics.

Table 4.9 shows the results of *t*-test and descriptive statistics of FSE, OS and PST by students' Y12 performance in chemistry for categories *A* and *B*. Students with Y12 performance in chemistry for category *A* ( $n = 138$ ) scored as follows for OS:  $M = 63.92$  ( $SD = 10.61$ ). By comparison, students' Y12 performance in chemistry for

category *B* ( $N = 114$ ) scored as follows for OS:  $M = 59.13$  ( $SD = 12.27$ ). To test the hypothesis that students' Y12 performance in chemistry for categories *A* and *B* were associated with statistically significantly different mean in OS, an independent sample *t*-test was performed. Distributions for *A* and *B* were sufficiently normal for the purposes of conducting a *t*-test (i.e., skew  $< |2.0|$  and kurtosis  $< |9.0|$ ) (Schmider et al., 2010). Additionally, the assumption of homogeneity of variances was tested and satisfied via Levene's *F* test,  $F(250) = 0.854$ ,  $p = 0.356$ . The independent sample *t*-test was associated with a statistically significant effect,  $t(250) = 3.33$ ,  $p = 0.001$ . Thus, students who scored highly in Y12 chemistry had a statistically significantly larger mean in OS than those who scored less.

Table 4.9

*Results of t-Test and Descriptive Statistics of FSE, OS and Test by Students' Y12 Performance in Chemistry for Categories A and B*

Student Performance	Average Y12 Chemistry Score						95% CI for Mean		
	A			B			Difference	t	Df
M	SD	n	M	SD	n				
FSE	57.86	13.85	139	54.71	14.28	120	[-.29,6.59]	1.80	257
OS	63.92	10.61	138	59.13	12.27	114	[1.95,7.64]	3.33*	250
PST	55.62	17.33	122	50.62	14.62	118	[.91,9.08]	2.41*	238

Note: \* $p < 0.05$ ,  $n=327$ . Total student participants in test and survey. Missing data were omitted.

From Table 4.9, we can observe that students with better scores in Y12 chemistry were associated with a statistically significantly larger mean in the PST than those with lower scores in Y12 chemistry. By comparing the means of FSE between these two categories of students, the results from Table 4.9 show that Y12 chemistry scores are not statistically significant in terms of contribution to success in the PST. The correlational analysis in Chapter 6 will further confirm these results.

### 4.3.2.3 Programming environments used in CS1

Question 6 asked students to select the programming environments used by lecturers to teach CS1. The results show that the most commonly used programming environments by lecturers was Turbo C++ (n = 124), followed by Microsoft Visual Studio (n = 76), terminal/command line (n = 33) and Dev-C++ (n = 11)

Question 7 asked students whether the programming environment used by their lecturer in CS1 was easy to use; 111 students of 124 said that it was easy to use Turbo C++, 68 students of 76 said that it was easy to use Microsoft Visual Studio, 30 students of 33 said that it was easy to use the terminal/command line and 10 students of 11 said that it was easy to use Dev-C++. The remaining students stated that the programming tools used by their lecturers were not easy to use; 13 students of 124 said that it was not easy to use Turbo C++:

*S039: Poor technology background*

*S156: Difficult to understand the interface and felt bit uninteresting*

*S217: Did not get much time to learn the interface*

Figure 4.2 shows the interface and screen of Turbo C++.

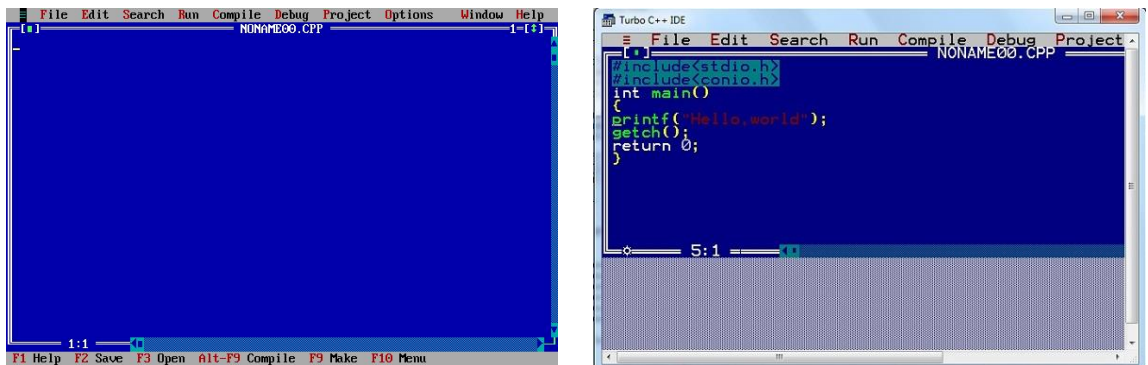
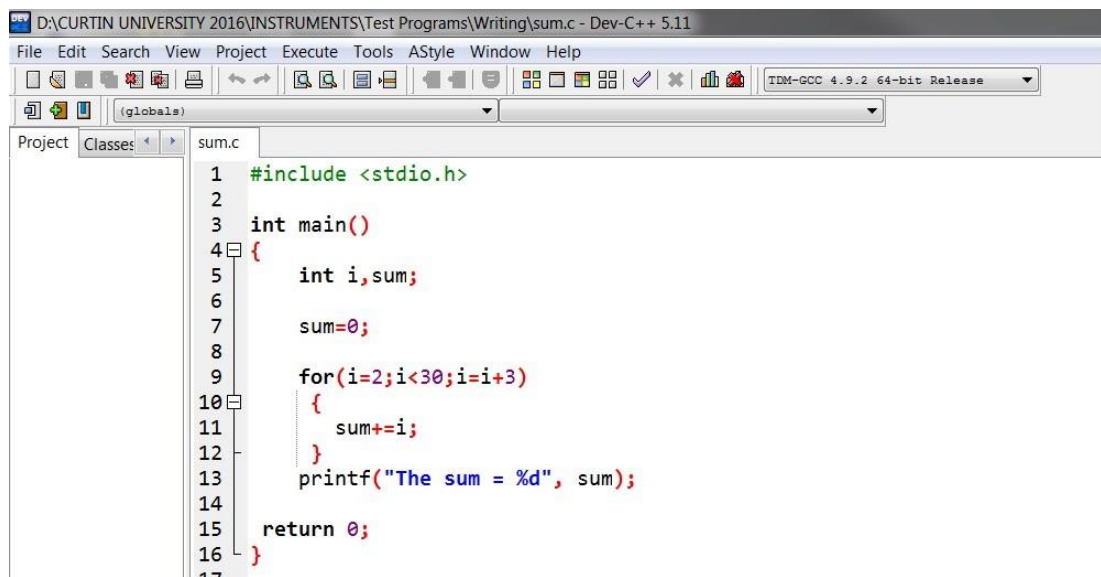


Figure 4.2. Interface and screen of Turbo C++.

We can conclude from the response that some students found it difficult to understand the interface of Turbo C++ and did not like the colour of the screen. The default colour is blue, as shown in Figure 4.2.

The interface of Dev-C++ is shown in Figure 4.3. Only one student of 11 said that it was not easy to use Dev-C++ but opted not to state the reason. The remaining 10 students found Dev-C++ user-friendly, which means that the interface was easy to use and understand, and not difficult to learn. The sample programs reported in this thesis were written, compiled and run in Dev-C++ by the researcher. The researcher finds Dev-C++ easy to use and simple to install; moreover, it is free.



```
D:\CURTIN UNIVERSITY 2016\INSTRUMENTS\Test Programs\Writing\sum.c - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes sum.c
1 #include <stdio.h>
2
3 int main()
4 {
5     int i,sum;
6
7     sum=0;
8
9     for(i=2;i<30;i=i+3)
10    {
11        sum+=i;
12    }
13    printf("The sum = %d", sum);
14
15    return 0;
16 }
```

Figure 4.3. Interface of Dev-C++.

Three students of 33 said that it was not easy to use terminal/command line. Only one student stated their reason, but this reason was found not to be relevant. Figure 4.4 shows how to compile C programs using the command line in windows. The command to compile C program in the Linux operating system is the same.

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\Users\Admin>cd Desktop
C:\Users\Admin\Desktop>gcc first.c
C:\Users\Admin\Desktop>first
Hello World
C:\Users\Admin\Desktop>
```

Figure 4.4. Command to compile and run C program using command line in windows.

Eight students of 76 said that it was not easy to use Microsoft Visual Studio.

Two students provided the following reasons:

*S178: Hard time to deal with software*

*S181: Software not user-friendly*

Figure 4.5 shows the interface of Microsoft Visual Studio.

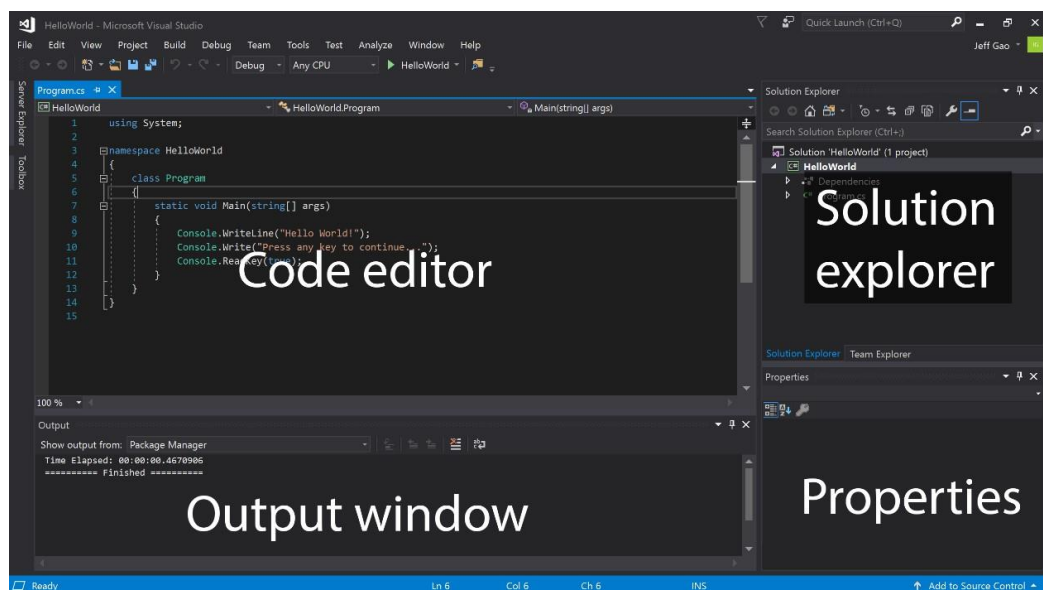


Figure 4.5. Interface of Microsoft Visual Studio

Source: ("Get started with Visual Studio 2017,")

Question 8 asked students if they had experienced any other easy-to-use programming environments apart from those their lecturer used in CS1. Eight students of 277 explored other programming tools, such as **PyCharm** and **Code::Blocks** and stated that these were easy to use. The lecturer could try these programming environments and determine if they are simple to use for beginners.

#### ***4.3.2.4 First programming language to be taught in CS1***

Question 9 asked students to give their opinion based on their experience in regard to the first programming language to be taught in CS1. Table 4.10 outlines the programming language students asserted should be taught in CS1; 59 percent of student nominated C as the first programming language that should be taught in CS1. This is probably because they have learnt C in CS1. Students who responded with C++, Java, Python, VB.Net and C# might have had some experience prior to CS1.

Table 4.10

*Number of Students' Response to First Programming Language to be Taught in CS1*

*(n = 277)*

Programming Language	n	Percent
C	164	59%
C++	73	26%
Java	57	21%
Python	18	7%
VB.Net	6	2%
C#	4	1%

Note: Participants could select more than one response.

Question 10 asked students to provide reasons for selecting a particular language to be taught in CS1. Students stated:

*S004: It [C] is the fundamental of any programming language.*

*S009: It [C] is easier compared to other languages like Java, C++, etc.*

*S008: Because of curiosity and interest in that language [C++].*

*S024: Some of us already learnt Java in class 12, so I think it will be easier for us too [Java].*

*S038: I heard that it is an interesting programming language [Java].*

Students' comments on the first programming language to be taught in CS1 shows their understanding of C as a foundation language that is easy to learn compared to other programming languages. Some students with experience in Java wanted to continue with Java in CS1, believing it would be easier for them to learn. Other students who responded with C++ and Java were curious to explore these languages.

#### ***4.3.2.5 Teaching/learning methods and practices that have helped in the learning of CS1***

Question 11 asked students to indicate the teaching/learning methods and practices they had experienced that helped them learn in CS1. Table 4.11 shows the teaching/learning methods and practices that students experienced in CS1 that helped them learn. Of the four teaching/learning practices and methods listed in Table 4.11, live coding by the lecturer (i.e., when lecturers teach by demonstrating the program from scratch using the programming environment in the class rather than explaining a static code from the slide) was found to be the most helpful in learning CS1 (60 percent) followed by reading the materials online (49 percent), pair/group programming (i.e., working in pairs and groups: 46 percent) and watching YouTube tutorials (20 percent).

Table 4.11

*Teaching/Learning Methods (n = 277)*

Teaching/ Learning Methods and Practices	n	Percent
Live coding by lecturers	168	60%
Reading online materials	135	49%
Pair/group programming	128	46%
YouTube tutorials	57	20%

Note: Participants could select more than one response and list their own. Reading materials online and YouTube tutorials were listed.

***4.3.2.6 Students' perceptions of the order of programming skills to be learnt in CS1***

Question 12 asked students to order the five programming skills based on their experience in learning CS1 from 1 (first programming skill learnt) to 5 (last programming skill learnt). Students were also asked to write the same number on the programming skills if they felt that some of the programming skills could be learnt in parallel. Table 4.11 presents the results based students' experience of learning in CS1. The results indicate that algorithm design (2.08) was the skill that the students generally suggested to begin with, followed by explaining (2.3) and translating (2.58). We can also surmise that students believed these three skills may be learnt at similar times, as the difference in means was low. Similarly, writing (3.0) and tracing (3.58) can be learnt at similar times, as there was little difference in their means. Students' interview responses to this question (see Section 4.4.2) will further provide information on this.



Table 4.12

*Summary Statistics of the Order of Programming Skills to be Learnt in CS1*

(*n* = 277)

Programming Skills	Mean	Std. Deviation
Tracing	3.58	1.31
Writing	3.00	1.33
Translating	2.58	1.01
Explaining	2.30	1.43
Algorithm design	2.08	1.19

*Note.* n=272. Some participants did not respond.

***4.3.2.7 Students' perception of the order of programming skills in terms of contribution to student performance in CS1***

Question 13 asked students to order the five programming skills based on their experience in learning CS1 to student performance in CS1 from 5 (highest) to 1 (lowest). Students were also asked to write the same number on the programming skills if they believed some of the programming skills were equally associated to their success in CS1. Table 4.13 presents the results based on students' experience in learning in CS1. The results show that writing contributes the most (3.14) to student performance, followed by explaining (2.99), tracing (2.97), algorithm design (2.86) and translating (2.73).

We can also say that explaining, tracing, algorithm design and translating are perceived by students to be approximately equally associated with success in CS1 after writing, as their means were not statistically significant. The path analysis in Chapter 6 will further examine the hierarchy among the five programming skills.

Table 4.13

*Summary Statistics of the Order of Programming Skills in Terms of Contribution to Student Performance in CS1 (n = 277)*

Programming Skills	Mean	Std. Deviation
Writing	3.14	1.40
Explaining	2.99	1.53
Tracing	2.97	1.43
Algorithm design	2.86	1.43
Translating	2.73	1.19

Note:  $n = 267$ . Some participants did not respond.

#### ***4.3.2.8 Approaches to learning in CS1***

Question 15 asked students to indicate their approach in learning CS1 by answering 20 items scored on a 5-point Likert scale of Biggs's (R-SPQ-2F). Questionnaire items were categorised into two learning approaches: deep and surface. There are two subscales of deep approach (deep motive and deep strategy) and two subscales of surface approach (surface motive and surface strategy). Table 4.14 shows the summarised statistics scores of the completed Biggs questionnaire. The mean and standard deviation show that students are not particularly aligned with either deep or surface learning approaches. See Section 6.2.5 for correlation of learning approach with student performance.

Table 4.14

*Aggregate Statistics on Students' Learning Approach Using Biggs Questionnaire*

(*n* = 277)

Learning Approach (Total Score)	Mean	Standard Deviation
Deep approach (50)	27.56	5.45
Deep motive (25)	13.00	3.27
Deep strategy (25)	14.53	2.99
Surface approach (50)	27.96	5.94
Surface motive (25)	13.76	3.23
Surface strategy (25)	14.20	3.49

Note: *n* = 275. Some participants did not respond.

#### **4.4 Student Qualitative Results**

This sections reports on the results of student qualitative data from the survey, and individual and group interviews. The qualitative data were classified by examining the number of occurrences of key terms. Results were summarised.

##### **4.4.1 Student qualitative survey results**

Question 14 of the survey asked students to suggest how we could improve teaching/learning of CS1 at RUB. Students' response to this questions were classified based on the number of occurrences of the particular areas:

##### **a) Same tutor for both theory and practical session**

Students stressed that it was important for them to have the same tutor for both theory and practical sessions. At CST, theory classes were taken by one lecturer and practical classes by another lecturer. Student responses were:

*S043: Better to have same tutor for both theory and practical sessions.*

*S208: Theory and practical should be taught by the same teacher. Moreover, things taught in the class should be practically close in the lab.*

*S056: Same tutor for both theory class and practical classes.*

It is true that the best scenario would be to have the same tutor for theory and practical sessions.

**b) Theory and practical classes in parallel**

Students also stressed about having theory and practical classes in parallel and not on different days. Only JNEC offered theory and practical classes in parallel in the computer laboratory. SC and CST have theory classes for one hour every day for four days and practical classes for three consecutive hours a week. A sample of students' responses were:

*S066: Better if the teachers teach the theory and practicals in a parallel system so that each students can follow.*

*S127: Theory and practical should run parallel.*

*S085: If we are allowed to bring our personal laptops in the lecture and practice parallel with what the tutor is teaching in the class, so that students may be able to perform better.*

Students' suggestions for conducting theory and practical classes together in computer laboratory does not seem feasible in the current situation. Currently, at SC and CST, computer laboratory are shared among students enrolled in programs with practical components. However, the researcher believes that allowing students to bring their laptops to the theory class should not be a problem.

**c) Experience lecturer in CS1**

Students strongly suggested that experienced lecturers should be assigned to teach CS1. This seems true, as based on the researcher's experience teaching a

module whose prerequisite is CS1, extra time needs to be spent explaining the basic concepts of programming, which students were supposed to learn in CS1.

A sample of students' responses were:

*S139: Experienced teacher should teach CS1.*

*S245: If an experienced teacher teaching CS1 is assigned for teaching, students can learn more and understand the concepts clearly.*

*S234: Experienced teacher who knows programming well and with high-level communication skills.*

*S205: By sending experienced tutors, as some of the tutors are not sure about the topics themselves.*

*S080: Require more experienced teacher teaching this module.*

*S087: Tutors should seek help from seniors to teach CS1.*

*S260: Students will learn maximum if the teacher with more experience and good teaching technique is given, as the programming is complex and logical compared to others.*

#### **d) Live coding and pair programming**

Students suggested lecturers should demonstrate programs live from scratch using programming environments when teaching CS1 classes and encourage students to work in pairs. A sample of students' responses were:

*S019: Live coding by the tutor is very useful.*

*S024: Supervised pair-learning and live coding with the teacher in the class.*

*S184: The tutor can write two to three lines of codes and explain to the students how that particular code works and let the students write their own code for a particular program.*

*S158: Flow of the program should be made clear to the students.*

#### **4.4.2 Student interview results**

This section reports on the qualitative data collected from individual and group interviews; 23 and 24 students participated in individual and group interviews respectively. Table 4.15 presents the details of students from seven programs who participated in individual interviews. Table 4.16 shows the number of students who participated in a group interview at the three colleges of RUB.

Table 4.15

*Programme and Number of Student Participants in Individual Interviews*

College	Programme	Male	Female	Total
CST	B. Eng. in Electronics and Communication	1	2	3
	B. Eng. in Electrical	2	1	3
	Bachelor of Architecture	1	3	4
	B. Eng. in Information Technology	3	0	3
	B. Eng. in Civil	1	2	3
JNEC	Diploma in Computer Hardware and Networking	2	2	4
SC	B.Sc. Computer Science	2	1	3
Total students		12	11	23

Table 4.16

*College and Number of Student Participants in Group Interviews*

College	Male	Female	Total
CST Group 1	3	2	5
CST Group 2	5	3	8
JNEC	3	3	6
SC	3	2	5
Total students	14	10	24

The areas covered in interviews were similar to those covered in the survey (see Section 4.3). The semi-structured interview questions were utilised but the order of the questions asked and the resulting conversations were not controlled. The following is a summary report of students' responses to the areas covered in individual and group interviews.

***4.4.2.1 Prior computing experience***

Students with non-programming computer experience such as information searches using the internet, computer games and application software such as Office stated that experience in these activities did not help them learn in CS1. Three students of 23 in individual interviews stated that the experience in searching for information using the Internet and Office usage helped them learn in CS1. According to Student S219, experience in Microsoft Word helped him type programs faster. Students S077 and S215 stated that knowing how to search for information using Google helped them find information related to topics in CS1 that they found difficult:

*S219: I feel it helps in typing the program faster.*

*S077: Knowing how to Google helped me to find information on topics that I find difficult in CS1.*

*S215: Knowing how to research information helped in learning CS1.*

The student with some programming experience stated that they understood the concepts in CS1 classes easier than their friends, who had no experience in any programming language. A total of 21 of 23 students stated that it would be better if students had some programming experience prior to taking CS1. All students in a group interview collectively said that they required programming experience prior to taking CS1 classes. Group 4 stated:

*G4: Those students who had previous knowledge on programming did pretty well in CS1 while we struggled.*

This results confirms the quantitative survey results reported in Section 4.3.2, in which the average mean of students' performance in CS1 having prior programming experience was higher than for students without programming experience.

#### ***4.4.2.2 Y12 performance in mathematics, physics and chemistry***

More than 50 percent of the students interviewed stated that good performance in mathematics is required, while performance in physics and chemistry is not necessary prior to learning CS1:

*S274: For writing programs on area, perimeter, factorials, Fibonacci numbers, etc., we need mathematics background.*

*S125: Mathematics logic is required.*

*S077: I don't think we require chemistry background. Maybe physics.*

Students' interview responses align with the quantitative survey results reported previously—students who scored highly in Y12 mathematics performed well in CS1.



#### ***4.4.2.3 Programming environments used in CS1***

A total of 11 of 23 students interviewed reported that they used Turbo C++ in CS1. More than half of the students who have learnt programming in CS1 using Turbo C++ stated that it is not user-friendly and not compatible with the operating system:

*S123: It is not easy to use.*

*S125: It is not compatible with the operating system. I cannot open the screen in full. Also, the software crashes in the middle of writing programs.*

*G01: Interface was not user-friendly.*

*G04: It wasn't very user-friendly. You know, whenever we open the screen, we cannot minimise and also we cannot copy and paste. I find it very difficult to use.*

These results do not align with the survey results discussed previously, in which few students (13 of 124) stated that Turbo C++ was not user-friendly.

Eight of 23 students' interviewed had used Microsoft Visual Studio in CS1. Over half of the students who learnt programming in CS1 using Microsoft Visual Studio stated that it is user-friendly:

*S071: Easy and user-friendly.*

*S018: It was not easy to use initially but later on I got used to it.*

These results align with the survey results discussed previously, in which only eight of 76 students stated that Microsoft Visual Studio was not user-friendly. The remaining students found it easy to use.

Only two of 23 students' interviewed had used terminal/command line in CS1. Out of two, only one student found it difficult to use:

*S274: Bit difficult. Need to remember all the commands.*

These results align with the survey results discussed previously, in which few students (three of 33) stated that using terminal/command-line was easy. The remaining students found it difficult to remember the commands.

Students who had explored other programming tools, such as PyCharm and Dev C++, stated that these were user-friendly.

*S127: I used PyCharm. It was easy to use and also more efficient.*

*S125: PyCharm. It is more user-friendly. I prefer PyCharm, as it is easier to use than Turbo C++.*

*S123: I used Dev C++. It is user-friendly. I prefer Dev C++ to Turbo C++, as Dev C++ is more user-friendly.*

The students' interview responses confirmed that students had explored other programming environments and found them easy to use. Thus, it is worth exploring PyCharm and Dev C++ and their suitability for use in CS1.

#### ***4.4.2.4 First programming language to be taught in CS1***

More than half of the students stated that CS1 should be taught using C as the programming language, while two students of 23 stated that Python should be the first programming language learnt:

*S127: I heard that Python is simple and basic so I guess Python can be used in CS1.*

*S125: I heard that Python is simpler than C, so maybe Python can be used to teach beginners.*

All the students in a group interview stated that the first programming language learnt should be C:

*G01: C is the foundation of other languages.*

*G03: Should be taught for beginners, as we are learning C++ in the second semester.*

#### ***4.4.2.5 Teaching/learning methods and practices that have helped in the learning of CS1***

All 23 students interviewed stated that working with pairs or in groups and live coding by lecturers helped them learn in CS1:

*S127: Working with friends helps in debugging the code [group programming].*

*S123: I can understand better from peers [pair/group programming].*

*S125: Working in groups helps us in sharing and exchanging knowledge and ideas [group programming].*

*S154: Working with friends helps in explaining to each other. Sometimes friends understand the concepts better than you [pair programming].*

*S010: Makes it easier to understand [live coding].*

*G4: It helps especially in finding errors [live coding].*

Students' responses in interviews and surveys indicate that live coding by lecturers and working in pairs/groups seems to benefit students in learning CS1.

#### ***4.4.2.6 Students' perceptions of the order of programming skills to be learnt in CS1***

All students interviewed (individual and group) stated that learning CS1 should begin with algorithm design before progressing to the next programming skills. Most students interviewed gave the preferred order of programming skills (begins with 1) to be learnt in CS1 as:

1. algorithm design
2. translating
3. writing

4. tracing
5. explaining.

This was compared with the order reported in the quantitative results (mean value shown in parentheses). The lower the mean value, the earlier the skills that students stated to be learnt first.

1. algorithm design (2.08)
2. explaining (2.30)
3. translating (2.58)
4. writing (3.00)
5. tracing (3.58).

Explaining was perceived to be the last programming skill to be learnt (according to interview results) while tracing was perceived to be the last programming skill to be learnt (from survey results). Both interview and survey results stated that student should start with algorithm design first and follow with the other programming skills in no particular order.

#### ***4.4.2.7 Students' perception of the order of programming skills in terms of their contribution to the Student Performance in CS1***

A total of 12 students of 23 stated that writing contributes more to student performance in CS1, while six of 23 cited algorithm design as contributing highly to student performance. Others identified either explaining or translating as contributors to performance:

*S127: I think writing [writing] should be contributing more to the success of CS1. Because if you know how to write code, then tracing, translating and explaining becomes easier to understand.*

*S125: Writing [writing], as most of the questions in the exam were based on writing.*

*S134: I feel if students know how to write algorithm [algorithm design] and represent the logic in flowcharts, then obviously students can write the program.*

*S274: If the algorithm [algorithm design] is wrong, then the whole program is wrong.*

Compared to survey results, writing was reported as having the highest contribution to student performance in CS1 (3.14), which aligns with the interview response. Algorithm design was reported as fourth in the order in the survey (2.86), not second, as stated in an interview. The path analysis in the Chapter 6 will further confirm this order of programming skills in terms of their contribution to student performance in CS1.

#### ***4.4.2.8 Suggestions on how to improve teaching/learning of CS1 at RUB***

Individual and group students' interview responses to this question emphasised four key areas: same tutor for both theory and practical, prior programming experience, pair/group programming and live coding.

##### **a) Same tutor for both theory and practical sessions**

From the student interviews, the researcher discovered that in CST, they have two lectures assigned each for theory and practical sessions in the computer laboratory. At SC and JNEC, only one lecturer was assigned to teach CS1 for both theory and practical. Students who had one lecturer for theory and one for practical sessions strongly recommended assigning only one lecturer to teach CS1, as they found it difficult to follow:

*S191: Same tutor should teach the class. We had theory classes taught by one tutor and practical classes taught by another. It was difficult to follow.*

**b) Prior programming experience**

Students in individual and group interviews said that it would benefit them in the learning of CS1 if one programming subject was introduced for students taking science as their major in Y12 to become familiar with programming concepts:

*S125: Computer science should be introduced in high school at basic level.*

*G2: If the Ministry of Education can introduce some sort of programming courses in the primary and the high school, it would be better, as students will be familiarised with some programming concepts when they enrol in colleges and study CS1.*

*G4: As a beginner to computer programming, I did not really understand what C programming was in the first few days of class. I feel it is important to introduce one programming subject to all the science students in Y12 so they are not lost in the class during CS1 classes.*

This qualitative result aligns with the quantitative results—the mean of student performance for those with prior programming experience was higher than for those without prior experience. Thus, it is clear that students must be familiarised with some basic programming concepts in Y12 so they are not overwhelmed when introduced to CS1 in the first semester.

As for those with non-programming computer experience, most had prior experience in activities such as Office, playing games, browsing the Internet and using social media.

### c) **Live coding and pair/group learning**

The topics of live coding and pair/group learning/programming arose repeatedly in students' response to the open-ended question in the survey. Students interviewed also said that live coding by lecturers helped them learn how to write a program in the programming environment, debug and observe the output of the program. Similarly, students stated that they learnt more in pairs and groups, than they did alone.

*S134: Students should be grouped into pairs and let them explore the topics [pair programming].*

*S071: Tutors should allow students to bring their laptops to the class so students can do live coding [live coding] with the tutors instead of students watching tutors to demonstrate.*

*G4: I think to explain the concepts of programming well, the tutor should focus more on live coding [live coding].*

Table 4.17 compares students' open-ended survey questions and interview responses to the suggestion of how to improve teaching/learning of CS1 at RUB. Commonly mentioned topics in the survey and interviews were that the same tutor should conduct theory and practical sessions, and the benefits of live coding and pair programming. Conducting theory and practical sessions in parallel and the use of experienced lecturers to teach CS1 were issues raised in the survey. Students discussed requiring prior programming experience in the interview.

Table 4.17

*Comparison of Students' Survey Open-Ended Question and Interview Responses to the Suggestion of How to Improve Teaching/Learning of CS1 at RUB*

Survey	Interview
Same tutor for both theory and practical sessions	Same tutor for both theory and practical sessions
Live coding and pair programming	Live coding and pair/group learning
Theory and practical classes in parallel	Prior programming experience
Experienced lecturer for CS1	

#### ***4.4.2.9 Approaches to learning in CS1***

Students' interviewed were asked questions about their approaches to learning in CS1. The sample of students' responses that fell under the deep approach are shown below:

*S127: I practised the programs that were taught in the class. I browsed the Internet for more explanation and wrote more programs [deep approach].*

*S071: Research until I pretty much understood the topics and practice writing programs [deep approach].*

*S215: I referred books, asked friends, browsed internet, watched YouTube video tutorials and finally asked module tutor. I enjoyed CS1 classes and as a result I scored high marks as well [deep approach].*

*S243: Practise program codes. Revised the program taught in the class and lab. I tried different ways of writing the program to get the same solution. I got more practice by helping my friends. I can understand the concepts well and I was interested in learning computer programming [deep approach].*

Only one student interviewed fell under the surface approach:

*S134: I studied only the notes given out by the tutor [surface approach].*



*S10: I revised only the class notes given by the tutor [surface approach].*

Looking at the performance of these students in PST, FSE and OS, it is evident that students who employed deep approach of learning did well in CS1 as compared to those students who employed surface approach of learning. For instance, student S127 scored 68.67 in PST, 60 in FSE and 65 in OS; student S215 scored 66.33 in PST, 75 in FSE and 82 in OS; and student S243 scored 56 in PST, 57 in FSE and 59 in OS. Conversely, student S134 scored 29 in PST, 18.75 in FSE and 19 in OS; and student S10 scored 28 in PST, 30.50 in FSE and 50 in OS.

All the students in a group interview responded their approaches to learning CS1 which fell under the deep approach of learning. The responses are shown below:

*G1 and G3: Watch YouTube video tutorials, ask friends, browse Internet and ask tutor*

*G2: Browse Internet. Use mobile apps to learn C programming. Refer library books and ask friends.*

*G4: Practice and surf Internet for explanation and also ask friends. Refer lecture notes.*

## **4.5 Summary**

This chapter reported on the quantitative univariate results from the student PST and survey followed by qualitative results from the student survey and interviews.

It was reported that, on average, students enrolled in the architecture program performed well in the PST compared to students enrolled in other programs. On average, students performed well in algorithm design, translating, explaining and tracing followed by writing. The results reported were consistent with the pattern that would be expected. Overall, on average, the CST College performed well in the PST, followed by SC and JNEC.

It was reported that only 39 students of the 277 student participants had some computer programming experience prior to taking CS1. These students had experience in C, C++, Java, JavaScript and Python. The descriptive statistics showed that the students with prior programming experience performed better in CS1 across FSE, OS and PST. However, the independent sample *t*-test results showed that this was statistically significant for students' with prior programming experience in terms of OS only, not to FSE and PST. Similarly, the means of FSE, OS and the PST were all higher for students who spent less time playing computer games and all lower for students who spent more time playing computer games. However, the independent sample *t*-test results showed none of the non-programming computer activities were statistically significant.

The independent sample *t*-test results showed that students' Y12 performance in mathematics and physics were statistically significant when comparing means between groups for the FSE, OS and the PST. Students' Y12 performance in chemistry was found statistically significant only in terms of the difference between means between groups for OS and the PST, not to FSE.

The most commonly used programming environments used by the lecturers to teach CS1 at RUB was Turbo C++ and Microsoft Visual Studio. Some students also explored PyCharm and Code::Blocks. C programming language was reported as an ideal programming language to teach in CS1.

It was reported that teaching/learning methods and practices such as live coding, pair/group programming, reading materials online and watching YouTube tutorials would benefit student in learning CS1.

The order of programming skills to be learnt in CS1 and the order of programming skills in terms of their contribution to student performance was also

presented in this chapter. Students made some suggestions to improve the teaching/learning of CS1: The same tutor should teach both theory and practical classes, theory and practical classes should be taught in parallel, experienced lecturer should teach CS1, live coding and pair/group programming should be encouraged and students should have prior programming experience. The results from this chapter will be further discussed in the multivariate analysis in Chapter 6.

## **Chapter 5: Lecturer' Quantitative Univariate and Qualitative**

### **Results**

#### **5.1 Introduction**

Chapter 4 presented the students' quantitative univariate programming skills test (PST) and survey results, followed by students' qualitative results. This chapter presents the lecturers' quantitative univariate results followed by lecturers' qualitative results. The results of inter-rater reliability (IRR) between the researcher's marking and a second marker on sample scripts of students' PST are also presented in this chapter.

#### **5.2 Lecturers' Quantitative Univariate Results**

The total of eight lecturers, seven male and one female, from three Royal University of Bhutan (RUB) colleges participated in the survey. The only female lecturer participant was from College of Science and Technology (CST).

Survey data were collected using a printed questionnaire, which was given to the lecturer participants by the researcher. Once the survey was completed, it was handed back to the researcher. The areas covered in the lecturer survey questionnaire were very similar to those in the student survey (see Section 4.3.2): prior computing experience; Y12 performance in mathematics, physics and chemistry; programming environments used in CS1; first programming language to be taught in CS1, first programming paradigm; teaching/learning methods and practices that have improved students' learning in CS1; order of programming skills to be learnt in CS1; contribution of programming skills to student performance in CS1; and students' approaches to learning in CS1.

The survey questionnaire consisted of 15 questions in total. Questions were closed-ended, single and multiple responses and open-ended questions. Sections 5.2.1–5.2.8 report on the summary statistics of survey questionnaire.

### 5.2.1 Prior computing experience

As mentioned in Section 4.3.2, prior computing experience was categorised as prior programming and prior non-programming computer experiences. Question 1 asked the lecturer to indicate whether it is beneficial for students to have some programming experience prior to taking CS1 classes or not. Seven lecturers of eight stated that it is beneficial for students to have prior computer programming experience.

Question 2 asked the lecturer to select the prior programming language that would give students an advantage in CS1. Table 5.1 shows that all lecturer participants stated that experience in C would give students an advantage in CS1, followed by experience in Java and Python.

Table 5.1

*Lecturers' Response to Students' Prior Programming Language Experience (n=8)*

Programming Language	n
C	8
Java	5
Python	5
C++	2
C#	2

Note: Participants could select more than one response.

Question 3 asked lecturers to select prior non-programming computer experience that would assist students in CS1. From Table 5.2, we observed that most lecturers stated that prior experience using the Internet to search for information and use of application software would assist students in CS1.

Table 5.2

*Lecturers' Responses to Prior Experience in Non-Programming Computer Activities*

(n=8)

Non-Computer Programming Experience	n
Application software	7
Information search	6
Computer games	1

Note: Participants could select more than one response.

**5.2.2 Y12 performance in mathematics, physics and chemistry**

Question 4 asked lecturers to suggest the minimum Y12 scores required for mathematics, physics and chemistry. Table 5.3 reports the mean and standard deviation of students' Y12 scores in mathematics, physics and chemistry. From the table, we can observe that lecturer recommend that students have a good performance in mathematics (at least 62.86), followed by physics (57.86) and chemistry (55).

Table 5.3

*Summary Statistics of Students' Minimum Y12 Score in Mathematics, Physics and Chemistry, as Suggested by Lecturers*

Y12 Subject	n	Mean	Standard Deviation
Mathematics	7	62.86	7.56
Physics	7	57.86	6.99
Chemistry	7	55	5.00

The results presented here align with the student survey and interview data, in which students' performance in mathematics should be always higher than their performance in physics and chemistry.

### 5.2.3 Programming paradigm

Question 5 asked the lecturer to indicate the programming paradigm that is suitable for CS1. Although six lecturers of eight stated that a procedural programming paradigm suits CS1, two lecturer stated that object-oriented programming would suit CS1 and one lecturer stated that both paradigms would suit. Further discussion on this will be presented in Chapter 7.

### 5.2.4 Programming environments used in CS1

Question 6 asked lecturers to select the programming environment they have used to teach CS1. Table 5.4 reports on this data. The results show that the most commonly used programming environment by the lecturers to teach CS1 was Turbo C++, followed by Microsoft Visual Studio.

Table 5.4

*Programming Environment Used by Lecturers to Teach CS1 (n=8)*

Programming Tool	n
Turbo C++	5
Microsoft Visual Studio	4
Terminal/command line	1

Note: Participants could select more than one response.

The results presented in Table 5.4 confirm the results of the student surveys and interviews in regard to the programming environments used by lecturers to teach CS1: Turbo C++ and Microsoft Visual Studio.

Question 8 asked the lecturer to indicate an ideal programming environment to be used in CS1 based on their teaching experience, while Question 9 asked the lecturer to state the reason. Four of seven lecturers (one lecturer chose not to respond) chose Turbo C++ as an ideal programming tool to be used in CS1 for the following reasons:

*L01: Simple compiler with not many GUI [graphical user interface] buttons and options which would normally keep away from distraction. It has all that is required to get started with learning how to program.*

*L03: Turbo C++ is one of the oldest Borland compilers which lets programmers use all the functions unlike other compiler. Dev-C++ does not recognise a few header files that can be used in Turbo C++.*

*L08: Compared to Visual Studio, Turbo C++ lets students write code from scratch so it gives a platform for the students to practise code writing in detail. While in Visual Studio, many functions are in-built and getting autocorrected where the learning platform is minimal.*

Two lecturers out of eight chose using terminal/command line as an ideal programming tool to be used in CS1:

*L04: Since students are familiarised with Linux operating system, they can easily write and compile programs using basic text editor and command line tool. The students do not have to worry about the availability of the software.*

*L06: The environment seems user-friendly and easier for the students to use. They can compile, locate the errors easily and then run the program successfully*

Only one lecturer of seven chose Microsoft Visual Studio as an ideal programming tool to be used in CS1:

*L05: The environment seems user-friendly and easier for the students to use. They can compile, locate the errors easily and then run the program successfully.*

Although most lecturers identified Turbo C++ as the ideal programming tool for use in CS1, it is recommended (based on student surveys and interviews) to further



explore other programming tools, such as PyCharm, Dev C++ and Code::Blocks. Further discussion on this will be presented in Chapter 7.

### 5.2.5 First programming language to be taught in CS1

Question 10 asked lecturers to indicate the first programming language to be taught in CS1 based on their experience teaching CS1. Table 5.5 reports on the first programming language to be taught in CS1, as indicated by lecturers.

Table 5.5

*Lecturers' Response to the First Programming Language to be Taught in CS1 (n=8)*

Programming Language	n
C	8
C++	1
Java	1
Python	1
VB.Net	0
C#	0

Note: Participants could select more than one response.

All lecturers stated that C should be the first programming language to be taught in CS1:

*L01: Easy to learn, language is designed to be readable. Applications of these programming languages are relevant in scientific computing.*

*L03: C is a general-purpose, procedural and imperative language which is the basic of computer programming languages. Having foundation knowledge and skills in the above language can let students learn any other language.*

*L04: Since C language is a procedural language, students are given a platform to think about solving real-time examples and C is simple, thereby students see it as convenient to learn.*

*L06: C is popular and many languages are inspired from it. A student who has learnt C can move easily to other language. We teach object-oriented programming in second semester, so C is an ideal language for new students to learn.*

### **5.2.6 Teaching/learning methods and practices that best suit students in learning CS1**

Question 11 asked lecturers to indicate the teaching/learning methods and practices that best suit students in learning CS1. Of three teaching/learning methods and practices listed in Table 5.6, live coding by the lecturer was found to be highly suited for students in CS1, followed by working in pairs/groups. This aligns with the student survey and interview results discussed previously. Only one lecturer emphasised the importance of practising writing code independently.

Table 5.6

*Lecturer Response to Teaching/Learning Methods and Practices in CS1 (n=8)*

Teaching/Learning Methods and Practices	n	Percent
Live coding by the lecturers	7	87.5%
Pair/group programming	2	25%
Practise writing code independently	1	12.5%

Note: Participants could select more than one response and list their own. Practising writing code independently was listed.

### **5.2.7 Lecturers' perceptions of the order of programming skills to be taught in CS1**

Question 13 asked lecturers to order the five programming skills to be taught based on their experience teaching CS1 from 1 (first programming skill to teach) to 5 (last programming skill to teach). Lecturers were also asked to write the same number on the programming skills if they believe that some of the programming skills could be taught in parallel. According to Table 5.7, algorithm design (1.25) was the skill

suggested as the most appropriate to begin with, followed by translating (2.25), explaining (2.50), writing (3.13) and tracing (4.38). Lecturers believe that translating and explaining may be learnt at the same time, as the difference in means was low. Both lecturers and students suggested to begin with algorithm design and end with tracing. Refer to Section 4.2.3 for the student survey results.

Table 5.7

*Summary Statistics of the Order of Programming Skills to be Taught in CS1 (n = 8)*

Programming Skills	Mean	Std. Deviation
Tracing	4.38	1.06
Writing	3.13	1.13
Explaining	2.50	1.07
Translating	2.25	0.71
Algorithm design	1.25	0.46

### **5.2.8 Lecturers' perception of the order of programming skills in terms of their contribution to the Student Performance in CS1**

Question 14 asked lecturers to order the five programming skills in terms of their contribution to students' performance in CS1 from 5 (highest) to 1 (lowest). Lecturers were also asked to write the same number on the programming skills if they consider that some programming skills contribute equally to student performance in CS1. Table 5.8 shows that algorithm design (4) contributes the most to student performance, followed by translating (3.38) and writing (3.38), explaining (2.88) and tracing (2). The results presented in Table 5.8 do not align with the student survey results presented in Table 4.13, in which students perceived that writing contributes the most to student performance in CS1. The path analysis in Chapter 6 will further examine the order of the five programming skills in terms of their contribution to students' performance in CS1.

Table 5.8

*Summary Statistics on the Order of Programming Skills in Terms of Contribution to Student Performance in CS1 (n = 8)*

Programming Skills	Mean	Std. Deviation
Algorithm Design	4.00	1.41
Translating	3.38	0.92
Writing	3.38	0.92
Explaining	2.88	1.64
Tracing	2.00	1.19

### 5.3 Lecturer Qualitative Results

This sections reports on the results of the lecturers’ qualitative data from the survey and individual interviews. As mentioned in the student qualitative results, the lecturer qualitative data were also classified by examining the number of occurrences of key terms. Results were summarised and compared with the results presented previously in this chapter.

#### 5.3.1 Lecturer qualitative survey results

Question 15 asked lecturers to suggest how to improve teaching/learning of CS1 at RUB. Lecturers’ responses were classified similarly to the students’ responses to this question—based on the number of occurrences of the areas listed below:

##### a) Prior computer programming experience

Lecturers L2 and L5 suggested that students should have some computer programming experience in high school before taking CS1 in college:

*L2: Have at least some programming language concepts before they join college.*

*L5: The students should be taught some of the basics related to programming when they are in high school or lower secondary school.*

This point was not raised by the students in the survey, but most students interviewed stated that some programming experience prior to taking CS1 classes would be of benefit.

**b) Live coding**

Lecturer L3 raised the importance of illustrating programs (live coding) during teaching sessions to benefit students in the learning of CS1.

*L3: Teaching and learning through illustration of application of programming (real-time application). Live coding in laboratory teaching.*

**c) Conduct theory and practical classes in parallel**

Although lecturer L8 recommended conducting theory and practical sessions in parallel or without gaps to assist students in implementing the concepts faster, this was an issue due to the large number of students enrolled in one program. For instance, at CST, more than 100 students were enrolled in civil engineering in November 2016 and computer laboratory could accommodate only 30 students at a time.

*L8: If we conduct theory and practical together without gaps in between, students can implement the concept faster. But it is very difficult due to the large number of students.*

**d) Frequent review of CS1 module descriptor**

Lecturer recommended visiting the CS1 module descriptor as often as possible to update the content. However, he stated the RUB curricula review take place only after four years.

*L8: Reviewing CS1 module descriptor frequently (in short duration) may help students to learn in better ways with improvement of new technology. But RUB regulation allows to review only after four years.*

The comparison of student and lecturer responses to this open-ended question is shown in Table 5.9. Both students and lecturers suggests having theory and practical sessions in parallel and recommended the demonstration of programs by coding live in the class.

Table 5.9

*Comparison of Student and Lecturer Suggestions on How to Improve Teaching/Learning of CS1 at RUB*

Student Response	Lecturer Response
Theory and practical classes in parallel	Theory and practical classes in parallel
Live coding and pair programming	Live coding
Experienced lecturer in CS1	Prior programming experience
Same tutor for both theory and practical sessions	Frequent review of CS1 module descriptor sessions

### 5.3.2 Lecturer qualitative interview results

This section reports on the data collected from lecturer interviews. Eight lecturers participated in individual interviews. Table 5.10 presents the details of the lecturer participants from the three colleges of RUB.

Table 5.10

*Lecturer Interview Participants*

College	Male	Female	Total
CST	4	1	5
JNEC	1	0	1
SC	2	0	2
	7	1	8

The areas covered in the interview were similar to those covered in the survey (see Section 5.2). The semi-structured interview questions were utilised but the order of the questions asked and the conversation was not controlled. The following presents a summary report of lecturers' responses.

### ***5.3.2.1 Prior computing experience***

Lecturers L5 and L8 interviewed stated that it would benefit students to have some programming experience prior to taking CS1:

*L5: I think some kind of programming concepts should be introduced in high school. I feel if they have the concepts, it might help them to learn CS1 faster.*

*L8: Some kind of programming experience would be beneficial. I have observed that students who had some experience in programming understood the concepts in CS1 faster than the ones who had no prior experience.*

Similarly, lecturers L1 and L4 asserted that basic computer literacy was sufficient prior to taking CS1:

*L1: Basic literacy in IT would suffice. At least one can use computer, understands what a computer is and do a bit of word processing and spreadsheets should be fine to get started.*

*L4: A basic knowledge of Office should be sufficient.*

From the results of lecturer survey and interview, we can conclude that basic computer literacy should be introduced to students taking science in Y12, in addition to basic programming concepts.

### ***5.3.2.2 Y12 performance in mathematics, physics and chemistry***

As stated in the lecturer survey data and student interview data, the requirement to perform well in mathematics in Y12 stands out from the rest of the subjects, like physics and chemistry as L1, L4 and L6 pointed out:

*L1: I think so because mathematics is not just limited to calculations and numbers but students also learn logical reasoning while solving mathematical problems. I feel that students who have very sound background in mathematics may find it easy to learn CS1 as programming also involves logical reasoning.*

*L4: Mathematics helps students to gain brain analysing power.*

*L6: I have observed that students who scored well in their Y12 mathematics did pretty well in CS1.*

Lecturer L7 professed a different view. They said students who perform well in Y12 mathematics do not necessary perform well in CS1:

*L7: I think it depends. For example, students who did very well in Y12 mathematics have failed in CS1 and students who scored average marks in mathematics did well in CS1. So I think mathematics background is not really an indicator of success in CS1.*

The correlation analysis in Chapter 6 will further validate these results and indicate the extent to which mathematics performance is an indicator of success in CS1.

### **5.3.2.3 Programming paradigm**

Lecturers L4, L6 and L8 recommended that procedural programming paradigms be taught for beginners in CS1 for several reasons. They stated that it would be too much for students to learn the concepts of object-oriented programming, as they do not have any prior programming experience. Thus, starting with procedural programming would help them learn object-oriented programming easier:

*L4: If we teach object-oriented programming first, then it is like putting cart in front of the horse.*



*L6: Although object-oriented programming is useful in real-life application, students here did not have any prior programming experience so it would be too much for them to learn the concepts of object-oriented programming. If they have the concepts of procedural language then learning object-oriented programming would be easier.*

*L8: I think experience in procedural programming will help in learning object-oriented programming better as object-oriented programming is complex and might not be suitable for the beginners.*

#### **5.3.2.4 Programming environments used in CS1**

Half of the lecturers recommended Turbo C++ as an ideal programming environment to be used in CS1 because the interface is simple, there are limited distractions on the buttons and introducing fancy integrated development environments (IDE) may have a negative impact on students' learning of programming syntaxes and grammars, as the auto-complete function might not help students remember the syntax and grammar.

*L1: It is one of the preferred IDEs and a good platform to get started, as the interface is simple and there are not many distractions on the buttons. One thing I would like to mention especially when we are teaching the beginners, I think introducing fancy IDEs may also have some negative impact, as the auto-completion function does not help students remember the syntax and grammar.*

*L8: I feel using Turbo C++ will help the students to learn programming better than Microsoft Visual Studio, as this tool has code auto-completion functions, which will not allow the students to learn the language syntax and grammar.*

The remaining 50 percent of lecturers recommended Microsoft Visual Studio or terminal/command line as the ideal programming environments to be used in CS1.

Lecturer L6 stated that using Linux terminal/command line allows students to focus on learning the language rather than learning how to use IDE:

*L5: Both me and my students find it comfortable and user-friendly to use this [Microsoft Visual Studio] environment.*

*L6: At the beginning, students can just focus on learning the language rather than learning how to use IDE [terminal/command line].*

### **5.3.2.5 First programming language to be taught in CS1**

More than half the lecturers interviewed said that C should be the first programming language to teach in CS1, as it is the basic programming language. They believe that learning basic language would make learning any other language easier:

*L5: I believe C is the basic programming language and if students learn the basic language, then they can learn other language.*

Lecturer L1 suggested Python as the first programming language in CS1 because the language is simple and appropriate for the beginners:

*L1: I haven't tried myself but some of my friends who are experts commented that Python would be one language that has come out time and again because of its simplicity. So introducing beginning programmers using Python would be more appropriate. That's why we are suggesting Python to be used in our upcoming programmes.*

### **5.3.2.6 Teaching/learning methods and practices that best suit students learning CS1**

The lecturer responses covered three key areas: live coding, tutorial and group programming. Tutorial as a topic emerged from the lecturer interviews in regard to teaching/learning methods and practices that may assist students learning CS1. The

lecturer said that conducting tutorial classes once a week and giving one-on-one consultation benefits students learning CS1:

*L4: What I do whether in the class or in the computer lab, I take one problem and then show how we can solve that problem by writing an algorithm and drawing a flowchart. Then I translate them into programming codes and demonstrate how we can write programs, compile and run to get the outcomes. I also explain the outcome of the program. I think this methods helped students in learning CS1 [live coding].*

*L6: I used to explain the concepts first and give live demonstration of the programs and I asked students to do the same in the class [live coding].*

*L5: In the tutorial class, I give one-on-one consultation on the problems that students face. In each tutorial class, we have not more than 10 students and it is held once in a week. It helps both the tutor and the student to interact more. Moreover, in the practical class, students were given list of questions to solve [tutorial].*

*L6: I grouped students [group programming] and let them work on the given problem and let them demonstrate their program to the whole class. I feel that has helped students to understand the code better.*

The key points that arose from the student and lecturer survey and interview results were live coding and group programming. The one-on-one consultation was not pointed out by students.

#### ***5.3.2.7 Lecturers' perceptions of the best order of programming skills to be taught in CS1***

All lecturers and students interviewed stated the programming skill to begin with is algorithm design and the last is either tracing or explaining. All other

programming skills fall in between. Table 5.11 shows the order of programming skills to be learnt in CS1. Half of the lecturers interviewed listed the order as:

1. algorithm design
2. translating
3. writing
4. explaining
5. tracing

The remaining half of the lecturers interviewed listed the order as:

1. algorithm design
2. translating
3. writing
4. tracing
5. explaining

Table 5.11

*Comparison between Students' and Lecturers' Order of Programming Skills to be Learnt/Taught in CS1*

Four (50%) Lecturers	Four (50%) Lecturers	12 (52%) Student
1. Algorithm design	Algorithm design	Algorithm design
2. Translating	Translating	Translating
3. Writing	Writing	Writing
4. Explaining	Tracing	Tracing
5. Tracing	Explaining	Explaining

The order of programming skills presented in the first column of the table aligns with the order presented in the survey results, in which algorithm design (1.25) was suggested as the first skill and tracing was the last (4.38). Similarly, the order

presented in the second column (lecturer interview response) aligns with the order presented in the third column (student interview response).

### ***5.3.2.8 Lecturers' perception of the order of programming skills in terms of Contribution to Student Performance in CS1***

Although some lecturers interviewed said that writing contributes most to student performance in CS1, lecturers L1 and L5 stated that algorithm design contributes the most:

*L1: The basic foundation that I count should always attribute to the success (i.e., how you understand programming, so beginning with flowchart and writing an algorithm is the first thing). If you know that, then it can help student to develop program codes, so algorithm design should be the first priority.*

*L5: I think algorithm design is the foundation skill for the other skills. If students understand this skill, then the other skills should be easier to understand as well.*

Lecturers and students shared the same view—either writing or algorithm design contributes most to student performance in CS1. Lecturers and students said that writing contributes the most because if students are competent in writing programs, it means they are already competent in other programming skills. Similarly, both lecturers and students said that algorithm design contributes the most because it is a foundation skill. If student knows this skill well, other skills can follow, which ultimately contributes to success in CS1.

Therefore, we can report that both algorithm design and writing skills contribute the most to student performance in CS1. The path analysis in Chapter 6 will confirm this result.

### ***5.3.2.9 Suggestions for the improvement of teaching/learning of CS1 at RUB***

As mentioned previously, in student and lecturer surveys and student interviews, lecturers' responses to this question were classified according to the number of occurrences of the following key areas. Four key areas were identified: conduct theory and practical classes in parallel, teach CS1 using experienced lecturers, pair-learning and student motivation.

#### **a) Conduct theory and practical classes in parallel**

As stated in the student survey results, lecturer L8 believe it would benefit students in CS1 if both theory and practical classes ran in parallel. Lecturer L8 said that students might forget the concepts they learnt in the theory class due to the gaps between the theory and practical class:

*L8: Based on my experience, I feel that both theory and practical classes should be conducted at the same time in the computer lab. The current situation is that we have theory classes first in the classrooms and then after 3–4 days, we have practical class in the computer lab. By then, students forget the concepts because of the gap between the theory and practical. So I strongly recommend to take both theory and then practical at the same time in the computer lab.*

Therefore, the lecturer strongly recommended conducting both theory and practical classes at the same time in the computer laboratory. This issue will be further discussed in Chapter 7.

#### **b) Teaching of CS1 by an experienced lecturer**

As stated in the student survey results, lecturer L1 shared the same view that experienced lecturers should be assigned to teach CS1. He said it would be a

challenge for both the tutor and the student if the tutor begins teaching immediately after graduation. According to him, an experienced lecturer should teach so he can bring examples that can stimulate students' thinking:

*L1: To improve teaching and learning is to do with the teacher. The teacher should be experienced to some extent, that he can bring certain examples which can stimulate thinking among the students. That is very important. If the teacher himself just graduated and started to teach and has no experience, I think it will definitely challenge both the teacher and the student in bringing out the best application of what they are learning. I think that is more important in order to improve teaching and teaching of CS1 at RUB.*

**c) Group programming**

Both students and lecturers suggested group work to improve learning in CS1. Lecturer L6 recommended that other RUB colleges develop a programming club in which students from all semesters come together and learn programming; senior students can help junior students write programs:

*L7: Encourage students to work in groups.*

*L6: At CST, we have programming club where students do additional programming tasks and I would recommend other RUB colleges to do the same as well.*

**d) Motivation and interest**

One key area mentioned by lecturers (but not students) was motivation and interest. The lecturer stated that lecturers must be in a position to motivate students and students must take interest in the subject they are learning. Lecturer L1 stated that students should know the purpose of learning programming to gain motivation

throughout the programming journey. He also mentioned that students should be aware of their career prospects—what they aspire to do and why they are learning the module—to generate some kind of motivation to learn the module. It is useless to learn it as paper just to pass and obtain the degree, as they would not be able to do much in the application unless motivated to understand why they are learning:

*L1: If they do not have the motivation as to how they are going to use and where they are going to use, I think it would become very difficult for students to at least get started with their learning. So I think first, students should be sufficiently made aware about their career prospects and what they are going to do and why they are learning such topics, such languages and such tools in the class. If they know that, I think that would generate some kind of motivation to learn the module. Otherwise they will always learn it as a kind of paper just to pass and get the degree. Ultimately when it comes to application, I think they will not be able to do much unless if they were so motivated to understand why they are learning.*

*L5: First, the interest should come from the student. They should explore more on their own and not depend on their tutor.*

#### **5.3.2.10 Insights on how students in CS1 should approach their learning to achieve success**

Lecturers' response to this question provided four key areas: understanding the logical construct of the program, student interest, practice and exploration.

##### **a) Understand the logical construct of the program**

Two lecturers interviewed highlighted the importance of the student understanding the logical construct of the program. This can be achieved by representing the logic in the form of a flowchart and writing algorithms:



*L1: The logical construct of a program is very important to understand how the program works. To command the computer to do something, students should understand how they can command, so I believe the fundamental understanding of how the program run is by drawing a flowchart and writing an algorithm. I always give an example of making a cup of tea. Maybe there are different ways of making a tea but there are certain processes as to what you need first, what the ingredients are and what you will get at the end. There is no programming in there but it shows the logical flow of how the programming can be done.*

**b) Student interest**

According to Lecturers L3 and L6, students' interest in learning a particular subject seems to be another contributing factor to success in that subject. This is something to investigate further in the future, as it may be beyond the scope of this thesis:

*L3: The interest should come from the students to learn any subjects. Once you give your interest, I think success will be there with no doubt.*

*L6: Students should be passionate to learn and must willing to invest enough time to learn that particular concepts.*

**c) Practice**

Another important key point raised by Lecturers L5 and L6 was the importance of practise in writing programs to achieve success in CS1. They believe that programming language is like any other human language. To master it, we need to practise:

*L5: They should refer more books and practice more, browse Internet and moreover they can approach the tutor if they don't understand.*

*L6: Try to complete all the questions that are assigned during the practical class. Practise, practise, practise and keep on practising. C is a language as any other human languages, it can be mastered only through practise and repetition.*

#### **d) Exploration**

According to Lecturers L5 and L8, students should go beyond the classroom and explore on their own and not depend solely on the lecture notes. Students should explore the topics by referring to more books, practising more and reading online. Moreover, they should approach their tutor and friends if they do not understand:

*L5: It is the responsibility of every tutor to explain the concepts well, then students should not depend on the tutor, they should explore more on those topics. Exploring means they should refer more books, practise more, browse the Internet and moreover they can approach the tutor if they don't understand. Also ask their friends.*

*L8: Students should not depend only on the notes that lecturer has provided. They should explore more on their own and practice writing program codes.*

### **5.4 Inter-Rater Reliability**

IRR analysis was conducted to check the reliability of the PST results using the two-way mixed (instructor fixed and student random) model with absolute agreement (evaluating how close markers were in terms of their score). The output result shows that Instructor 2 (68.75) is stricter than Instructor 1 (70.16). By examining the average measures (0.964), there is high IRR between the two markers, as shown in Table 5.12.

Table 5.12

*IRR between Two Markers*

	Interclass	95% Confidence Interval		F Test with True Value 0			
	Correlation <sup>b</sup>	Lower Bound	Upper Bound	Value	df1	df2	Sig
Single Measures	.930 <sup>a</sup>	.85	.97	29.36	26	26	.000
Average Measures	.964 <sup>c</sup>	.92	.98	29.36	26	26	.000

Note: Two-way mixed effects model where people effects are random and measures effects are fixed.

a. The estimator is the same, whether the interaction effect is present or not.

b. Type A intraclass correlation coefficients using an absolute agreement definition.

c. This estimate is computed assuming the interaction effect is absent, because it is not estimate otherwise.

## 5.5 Summary

This chapter reported on the quantitative univariate results from the lecturer survey and qualitative results from the lecturer survey and interviews. This chapter also presented the IRR between the two markers of the students' PST.

The most commonly used programming environments by the lecturers to teach CS1 at RUB was Turbo C++ and Microsoft Visual Studio. C programming language was reported as an ideal programming language to teach in CS1.

It was reported that teaching/learning methods and practices such as live coding, pair/group programming, writing code independently would benefits student in learning CS1.

The lecturers' best order of programming skills to be learnt in CS1 and the order of programming skills in terms of their contribution to student performance was also presented in this chapter.

Lecturers made some suggestions to improve the teaching/learning of CS1: the same tutor should teach both theory and practical classes, theory and practical classes should be taught in parallel, experienced lecturer should teach CS1, live coding and

pair/group programming should be encouraged, students should have prior computer programming experience, students should be motivated and interested in learning, and one-on-one consultations in tutorial sessions should be considered.

The lecturer also provided some insights on how students in CS1 should approach their learning to achieve success in CS1. The results from this chapter will be further discussed in the multivariate analysis in Chapter 6 and 7.



## **Chapter 6: Bivariate and Multivariate Results**

### **6.1 Introduction**

Chapter 4 and 5 described the quantitative univariate results from the students' programming skills test (PST) and student and lecturer survey. It also reported on the data collected from student and lecturer interviews. This chapter presents quantitative bivariate and multivariate analysis of the students' PST and survey data. Data were analysed using simple correlations that describe the bivariate association between the variables. The values of correlations were calculated using bivariate Pearson correlation coefficients with a two-tailed test of significance. Data were then further analysed using multiple regression, which determines the predictors of student performance. In addition, path analysis was conducted.

The following sections begin with the presentation of the bivariate results from the students PST and survey data. They report on the association between student performance and: students' prior computing experience; students' Y12 performance in mathematics, physics and chemistry; students' performance in programming skills such as algorithm design, translating, tracing, explaining and writing; and students' learning approach. It also explores the association between programming skills.

Finally, it presents the results of linear multiple regression conducted to investigate the significant programming skill predictors that determine student performance in CS1, along with path diagrams constructed using the results of the multiple regression.

### **6.2 Student Bivariate Results**

This section presents the bivariate results of the student input factors, learning process factors and student performance.

### 6.2.1 Association between student performance and students' prior computing experience

As mentioned previously, prior computing experience was categorised as prior programming experience and prior non-programming computer experience. The univariate analysis in the previous chapter identified a statistically significant differences in means between students with and without prior programming experience to OS performance and no statistical difference in means between students with and without prior programming experience to FSE and PST scores (see Table 4.6). An examination of the simple correlation analysis shown in Table 6.1 indicates that an association between students' having prior programming experience and OS in CS1. Thus, this result seems to support the univariate results presented in Chapter 4, although the correlation was very low.

Table 6.1

*Association between Students' Prior Programming Experience and Student*

*Performance (n = 327)*

	FSE (n)	OS (n)	PST(n)
Programming experience	.072 (261)	.130* (254)	.077 (242)

Note: \* $p < 0.05$ .

Similarly, the univariate analysis for prior non-programming computer experience—such as information searches using the Internet and use application software such as Office and computer games—showed the difference in means of student performance with and without prior non-programming computer experience. However, it was not a statistically significant difference. An examination of the simple correlation analysis shown in Table 6.2 indicates a significant negative association between students who played computer games prior to taking CS1 and students'

performance in FSE in CS1. The remaining non-programming computer experience was not statistically significant to any of the student performance measures. The results indicate that only computer games seemed to have negative impact on students' performance in CS1. Prior experience in information searches and application software did not seem to have any impact on students' performance in CS1.

Table 6.2

*Association between Students who Played Computer Games and Student*

*Performance (n = 327)*

NPE	FSE (n)	OS (n)	PST (n)
Computer games	-0.140*(206)	-0.129(209)	0.022(193)

Note: \* $p < 0.05$ .

**6.2.2 Association between student performance and students' Y12 performance in mathematics, physics and chemistry**

Table 6.3 reports the results for the associations between student performance and Y12 performance in mathematics, physics and chemistry. The simple correlation analysis showed significant positive association between student performance and their Y12 performance in mathematics, physics and chemistry. While the results showed that students' Y12 performance in mathematics had a higher degree of association to FSE and OS, followed by physics and chemistry, students' Y12 performance in physics had a higher degree of association to the PST, followed by chemistry and mathematics.



Table 6.3

*Association between Student Performance and Y12 Performance in Mathematics, Physics and Chemistry (n = 327)*

Y12 Performance	FSE (n)	OS (n)	PST (n)
Mathematics	0.234**(257)	0.322**(251)	0.182**(238)
Physics	0.221**(257)	0.305**(251)	0.193**(238)
Chemistry	0.169**(259)	0.279**(252)	0.183**(240)

Note: \* $p < 0.05$ . \*\* $p < 0.01$ .

The univariate analysis results presented in Chapter 4 showed a statistically significant difference in means of student performance for students who scored highly in mathematics, physics and chemistry (see Table 4.7–Table 4.9)

It was confirmed from the univariate and bivariate correlation analysis that students' Y12 performance in mathematics, physics and chemistry did have some impact on student performance in CS1.

### **6.2.3 Association between students' ability in programming skills**

Table 6.4 shows the association among algorithm design, translating, tracing, explaining and writing. The simple correlation analysis indicates positive and significant associations among programming skills. Although the concepts of programming skills seem to develop in sequence—that is, they start with algorithm design and end with writing—skills like translating, tracing, explaining and writing overlap each other and develop simultaneously, so some correlation would be expected. As shown in Table 6.4, the degree of association between tracing and explaining is higher compared to others. This is followed by explaining and writing; translating and explaining, and translating and writing. This indicates that students competent in tracing a piece of code can explain what this code does in plain English and vice versa. Similarly, the ability to write code indicates that the student is

competent in explaining the purpose of the piece of code in plain English. The reverse may not be true all the time, as in explaining, the code is already given. The student just needs tracing skills to trace a piece of code and then explain the purpose in plain English. The association between translating and explaining indicates that students able to translate the algorithm design correctly into any high-level programming language can also explain in plain English what the translated code does and vice versa. The association between translating and writing indicates that students competent in translating the algorithm design correctly into any high-level programming language are also competent in writing codes. It seems true that the translated programs are closer to executable written programs.

Table 6.4

*Association among Algorithm Design, Translating, Tracing, Explaining and Writing*  
( $n = 292$ )

Programming Skills	Algorithm Design	Translating	Tracing	Explaining	Writing
Algorithm design		0.164** (289)	0.229** (286)	0.186** (285)	0.225** (276)
Translating			0.176** (287)	0.350** (286)	0.322** (277)
Tracing				0.366** (284)	0.215** (278)
Explaining					0.354** (277)
Writing					

Note: \*\* $p < 0.01$ .

#### **6.2.4 Association between student performance and students' ability in programming skills**

Table 6.5 shows the association between student performance and their ability in programming skills. The correlation results indicate a positive significant association between student performance and ability in algorithm design, translating, tracing, explaining and writing. The degree of association between student

performance and explaining was higher than the degree of association between student performance and the rest of the programming skills. This was followed by writing, translating, tracing and algorithm design. This may be suggestive of a possible hierarchy in the programming skills, which is further discussed in Section 6.3.

Table 6.5

*Association between Student Performance and Ability in Programming Skills (n = 327)*

Programming Skills	FSE (n)	OS (n)
Algorithm design	0.252**(283)	0.170**(219)
Translating	0.415**(284)	0.378**(220)
Tracing	0.353**(282)	0.304**(218)
Explaining	0.555**(280)	0.517**(219)
Writing	0.522**(272)	0.402**(213)

Note: \*\* $p < 0.01$ .

### **6.2.5 Association between student performance and students' learning approach**

As mentioned in Section 3.6.2, to examine how students' approach their learning in CS1, Biggs's R-SPQ-2F (which consists of 20 closed-response questions scored on a 5-point Likert scale) was chosen. The elements of the instrument were grouped under deep approach (deep motive and deep strategy) and surface approach (surface motive and surface strategy). Table 6.6 shows the association of students' learning approaches and their performance in CS1.

Table 6.6

*Association between Student Performance and Students' Learning Approach**(n = 327)*

Learning Approach	FSE (n)	OS (n)	PST (n)
Deep approach	0.096(252)	0.142*(245)	0.133(233)
Deep motive	0.084(252)	0.124(245)	0.101(233)
Question 4	0.156*(249)	0.160*(242)	0.077(231)
Deep strategy	0.082(252)	0.122(245)	0.130*(233)
Question 7	0.086(251)	0.126*(244)	0.100(232)
Question 8	0.147*(250)	0.165**(243)	0.192**(231)
Surface approach	-0.141*(251)	-0.135*(244)	-0.126(232)
Surface motive	-0.152*(251)	-0.144*(244)	-0.176**(232)
Question 12	-0.094(247)	-0.079(241)	-0.166*(228)
Question 13	-0.095(250)	-0.077(243)	-0.149*(231)
Question 15	-0.156*(251)	-0.159*(244)	-0.127(232)
Surface strategy	-0.0101(251)	-0.197(244)	-0.051(232)

Note: \* $p < 0.05$ . \*\* $p < 0.01$ .

The correlation analysis results show that both deep and surface approaches had significant correlation to student performance in CS1. That is, the deep approach was positively correlated to student performance and the surface approach was negatively correlated to student performance; these were the expected results. Although the correlations were significant, they were very low (0.142 [OS], -0.141 [FSE] and -0.135 [OS]). Some questions that significantly correlated to student performance was also shown in the table. The questions include:

*Question 4: I work hard at my studies because I find the material interesting.*

[Deep motive]

*Question 7: I find most new topics interesting and often spend extra time trying to obtain more information about them.* [Deep strategy]

*Question 8: I test myself on important topics until I understand them completely.* [Deep strategy]

*Question 12: I do not find my course very interesting so I keep my work to the minimum.* [Surface motive]

*Question 13: I find I can get by in most assessments by memorising key sections rather than trying to understand them.* [Surface motive]

*Question 15: I see no point in learning material which is not likely to be in the examination.* [Surface motive]

Although correlation coefficients of these questions to student performance were consistent (in terms of sign) with the expected outcomes from this instrument, they were very low. Therefore, we cannot conclude that students' learning approaches are determinants of student performance in CS1. This result may be attributed to the fact that English is not the students'/participants' first language, making it difficult for them to comprehend the questions in the questionnaire.

## **6.3 Multivariate Results**

In Chapter 4 and 5, lecturers and student were asked in the survey and interview about their perception of the order of programming skills in terms of their contribution to student performance in CS1. This section presents the relative contribution of the programming skills variables to student performance through multiple regression and path analysis to confirm/disconfirm the order of programming skills listed as perceived by the lecturers and students in Chapter 4 and 5.

### **6.3.1 Multiple regression analysis**

The first part of this section presents results of a multiple regression with students' performance in algorithm design, translating, tracing, explaining and writing as an independent variables (predictor variables) and FSE, OS and modified FSE

(MFSE) as dependent variables (output variables). The researcher realised that some questions in FSE were not related to the programming skill variables in this study. Thus, to improve regression model, the researcher removed questions that did not fall under algorithm design, translating, tracing, explaining and writing and recorded the new score as MFSE. The questions removed from FSE related to the theory of language features, such as defining ‘symbolic constant’ and declaring it; differentiating between ‘call by value’ and ‘call by reference’; writing notes on ‘C pre-processor’; explaining how to use ‘jump’ and ‘goto’ statements; where C programming language was developed; and questions on number systems conversion. The MFSE score for SC was 36, JNEC was 31.75 and CST was 39.5, which was out of 50 (SC), 40 (JNEC) and 50(CST) respectively. The CS1 FSE paper of SC, JNEC and CST is in Appendix L.

The second part of this section presents results of a multiple regression with students’ performance in algorithm design, translating, tracing and explaining as predictor variables and writing as the output variable.

***6.3.1.1 Multiple regression results with students’ performance in algorithm design, translating, tracing, explaining and writing as independent variables and FSE, OS and MFSE as dependent variables***

Table 6.7 shows the six assumptions that were checked before running multiple regression. All assumptions listed in Table 6.7 for the independent and dependent variables were met and statistical results are shown in Appendix I.

Table 6.7

*Verifying the Assumptions to Run Multiple Regression*

Assumptions	FSE	OS	MFSE
The relationship between the predictor variables and the output variable is linear	√	√	√
There is no multicollinearity in your data	√	√	√
The values of the residuals are independent	√	√	√
The variance of the residuals is constant (homoscedasticity)	√	√	√
The values of the residuals are normally distributed	√	√	√
There are no influential cases (significant outliers) biasing your model	√	√	√

Table 6.8 presents the results of the multiple regression analysis with student performance as the dependent variable and students' performance in algorithm design, translating, tracing, explaining and writing as independent variables.

Table 6.8

*Multiple Regression Results with Student Performance as Dependent Variable and Programming Skills as Independent Variables*

Variables	Student Performance					
	FSE		OS		MFSE	
	<b>B</b>	95% CI	<b>B</b>	95% CI	<b>B</b>	95% CI
Constant	27.63**	[22.70,32.55]	41.73**	[36.85,46.61]	19.53**	[14.26,24.80]
Algorithm design	0.036	[-0.024,0.096]	0.018	[-0.040,0.077]	0.018	[-0.045,0.082]
Translating	0.078**	[0.024,0.131]	0.066*	[0.016,0.117]	0.116**	[0.061,0.172]
Tracing	0.083**	[0.020,0.145]	0.068*	[0.011,0.126]	0.104**	[0.039,0.169]
Explaining	0.151**	[0.102,0.200]	0.126**	[0.079,0.173]	0.105**	[0.099,0.201]
Writing	0.220**	[0.151,0.290]	0.110**	[0.043,0.176]	0.234**	[0.162,0.307]
<b>R</b> <sup>2</sup>	0.454		0.365		0.484	
<b>F</b>	43.64**		23.701**		47.83**	

Note: \* $p < 0.05$ . \*\* $p < 0.01$ .

Multiple regression results with FSE as the dependent variable and algorithm design, translating, tracing, explaining and writing as independent variables indicates that the model explained 45 percent of the variance and that the model was a significant predictor of FSE,  $F(5,262) = 43.64, p = .000$ . translating ( $B = .078, p < .05$ ), tracing ( $B = .083, p < .05$ ), explaining ( $B = .151, p < .05$ ) and writing ( $B = .220, p < .05$ ) contributed significantly to the model, algorithm design ( $B = .036, p < .240$ ) did not. The final predictive model was:  $FSE = 27.63 + (.078 * translating) + (.083 * tracing) + (.151 * explaining) + (.220 * writing)$ .

Multiple regression results with OS as the dependent variables and algorithm design, translating, tracing, explaining and writing as independent variables indicates that the model explained 36.5 percent of the variance and that the model was a significant predictor of OS,  $F(5,206) = 23.70, p = .000$ . While translating ( $B = .066, p < .05$ ), tracing ( $B = .068, p < .05$ ), explaining ( $B = .126, p < .05$ ) and writing ( $B = .110, p < .05$ ) contributed significantly to the model, algorithm design ( $B = .018, p < .534$ ) did not. The final predictive model was:  $OS = 41.73 + (.066 * translating) + (.068 * tracing) + (.126 * explaining) + (.110 * writing)$ .

A multiple regression result with MFSE as the dependent variable and algorithm design, translating, tracing, explaining and writing as predictor variables indicates that the model explained 48.4 percent of the variance and that the model was a significant predictor of MFSE,  $F(5,255) = 47.825, p = .000$ . While translating ( $B = .116, p < .05$ ), tracing ( $B = .104, p < .05$ ), explaining ( $B = .150, p < .05$ ) and writing ( $B = .234, p < .05$ ) contributed significantly to the model, algorithm design ( $B = .018, p < .573$ ) did not. The final predictive model was:  $MFSE = 19.532 + (.116 * translating) + (.104 * tracing) + (.150 * explaining) + (.234 * writing)$ .



After running series of multiple regression, we can observe that the results using MFSE as an output variable is better in terms of variations explained by their independent variables compared to FSE and OS.

In each model, writing and explaining had the largest beta value and contributed most towards the predicted value of the dependent variable.

***6.3.1.2 Multiple regression results with students' performance in algorithm design, translating, tracing and explaining as independent variables and writing as the dependent variable***

At the end of one semester of CS1, students' were expected to write programs in any high-level computer programming language to solve problems. Also, since students' ability in writing represents the final programming skills for students in CS1, the researcher decided to use writing as a dependent variable and other programming skills variables—algorithm design, translating, tracing and explaining—as independent variables. Table 6.9 presents multiple regression results with writing as the dependent variable and algorithm design, translating, tracing and explaining as independent variables.

The results of the regression indicates that the model explained 18.8 percent of the variance in writing and the model was a significant predictor of writing,  $F(4,269) = 15.56, p = .000$ . While algorithm design ( $B = .120, p < .05$ ), translating ( $B = .158, p < .05$ ) and explaining ( $B = .163, p < .05$ ) contributed significantly to the model, tracing ( $B = .042, p < .444$ ) did not. The final predictive model was: writing = 13.94 + (.120 \* algorithm design) + (.158 \* translating) + (.163 \* explaining).

Table 6.9

*Multiple Regression Results with Writing as Dependent Variable and Other Programming Skills as Independent Variables*

Variables	Writing	
	<b>B</b>	95% CI
Constant	13.94**	[5.23,22.36]
Algorithm design	0.120*	[0.016,0.223]
Translating	0.158**	[0.068,0.248]
Tracing	0.042	[-0.066,0.150]
Explaining	0.163**	[0.081,0.245]
<b>R<sup>2</sup></b>	0.188	
<b>F</b>	15.56**	

Note: \* $p < 0.05$ . \*\* $p < 0.01$ .

Thus, algorithm design, translating and explaining are the significant predictors of writing which means students have to be competent in algorithm design, translating and explaining in order to perform well in writing. Although 18.8% is a relatively small amount of the variance in writing that is being accounted for by the other programming skills variables, it should be noted that this is only part of the final model that is developed in subsequent sections.

### 6.3.2 Path analysis

To better understand the multiple regression results shown above, the path diagram presented in this section was created to better visualise the relative contribution of the predictor variables to the outcome variables. Figure 6.1 shows the hypothesised causal ordering for how algorithm design, translating, tracing, explaining and writing predict student performance.

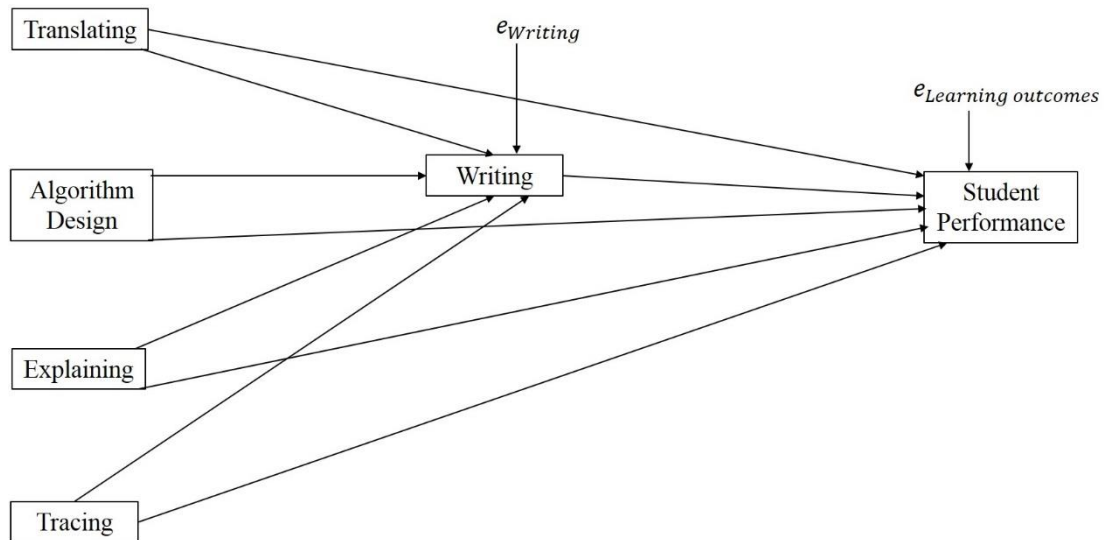


Figure 6.1. Hypothesised path model.

Figure 6.2—Figure 6.4 shows the path diagrams created with FSE, OS and MFSE as dependent variables and algorithm design, translating, tracing, explaining and writing as independent variables. Once the significant predictors of FSE, OS and MFSE were identified, the process was repeated with writing as the dependent variable and algorithm design, translating, tracing and explaining as independent variables.

In the path diagrams, variables are shown in the box, the beta coefficient with Pearson correlation in brackets is shown along the paths.  $R^2$  and error variance ( $e$  values) are shown underneath writing, FSE, OS and MFSE. Error variance values were computed using  $e = \sqrt{1 - R^2}$ .

As shown in Figure 6.2, students' marks on algorithm design questions accounted for only 4.7 percent of the variance in the marks scored on writing questions ( $R^2 = 0.047$ ). The marks students scored on translating questions accounted for only 15 percent of the variance in the marks scored on writing questions ( $R^2 = 0.153$ ) and the marks students scored on explaining questions accounted for only 13 percent of the variance in the marks scored on writing questions ( $R^2 = 0.13$ ). In combination, the

algorithm design, translating and explaining questions accounted for 19 percent of the variance in the marks scored on writing ( $R^2 = 0.188$ ).

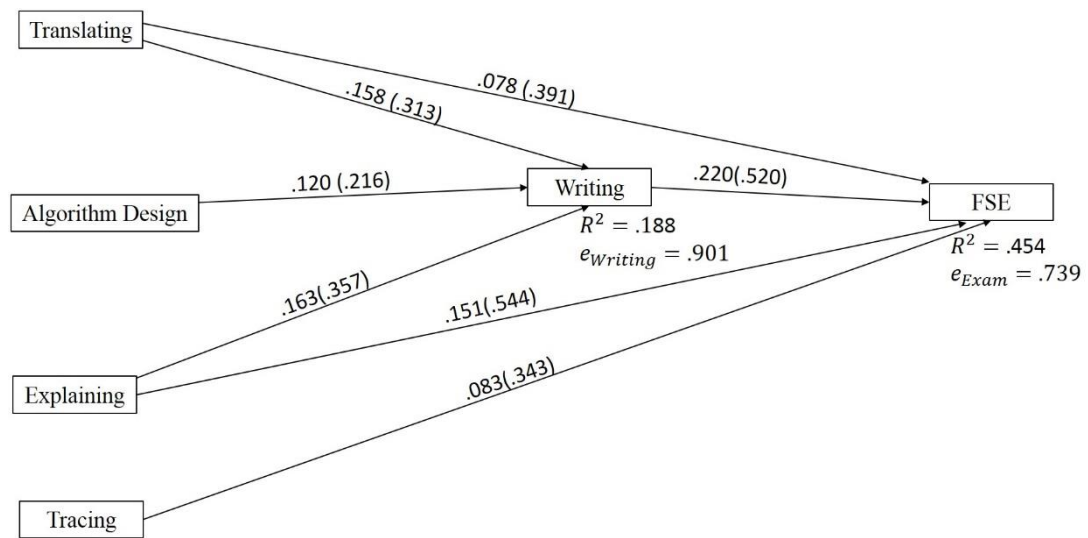


Figure 6.2. Path diagram with FSE and writing as output variables and programming skills as predictor variables.

Also, students' marks on translating questions accounted for only 15 percent of the variance in the marks scored on FSE ( $R^2 = 0.15$ ). Students' marks on tracing questions accounted for only 12 percent of the variance in the marks scored on FSE ( $R^2 = 0.12$ ). The marks students scored on explaining questions accounted for only 30 percent of the variance in the marks scored on FSE ( $R^2 = 0.30$ ) and students' marks on writing questions accounted for only 27 percent of the variance in the marks scored on FSE ( $R^2 = 0.27$ ). In combination, translating, tracing, explaining and writing questions accounted for 45 percent of the variance in the marks scored on FSE ( $R^2 = 0.454$ ).

Figure 6.3 indicates that students' marks on translating questions accounted for only 12 percent of the variance in the marks scored on OS ( $R^2 = 0.12$ ). Marks scored on tracing questions accounted for only 10 percent of the variance in the marks scored on OS ( $R^2 = 0.10$ ). Students' marks on explaining questions accounted for only 27

percent of the variance in the marks scored on OS ( $R^2 = 0.27$ ) and the marks scored on writing questions accounted for only 16 percent of the variance in the marks scored on OS ( $R^2 = 0.16$ ). In combination, translating, tracing, explaining and writing questions accounted for 36.5 percent of the variance in the marks scored on OS ( $R^2 = 0.365$ ).

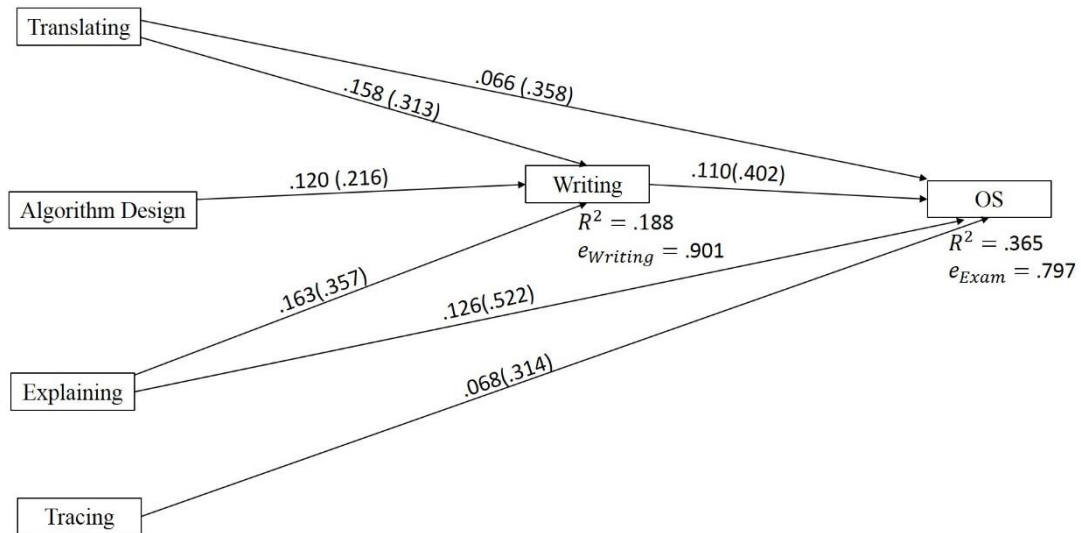


Figure 6.3. Path diagram with OS and writing as dependent variables and programming skills as independent variables.

Figure 6.4 indicates that students' marks on translating questions accounted for only 19 percent of the variance in the marks scored on MFSE ( $R^2 = 0.19$ ). Marks scored on tracing questions accounted for only 13 percent of the variance in the marks scored on MFSE ( $R^2 = 0.13$ ). Students' marks on explaining questions accounted for only 30 percent of the variance in the marks scored on MFSE ( $R^2 = 0.30$ ) and the marks students scored on writing questions accounted for only 28 percent of the variance in the marks scored on MFSE ( $R^2 = 0.28$ ). In combination, translating, tracing, explaining and writing questions accounted for 48 percent of the variance in the marks scored on MFSE ( $R^2 = 0.484$ ).

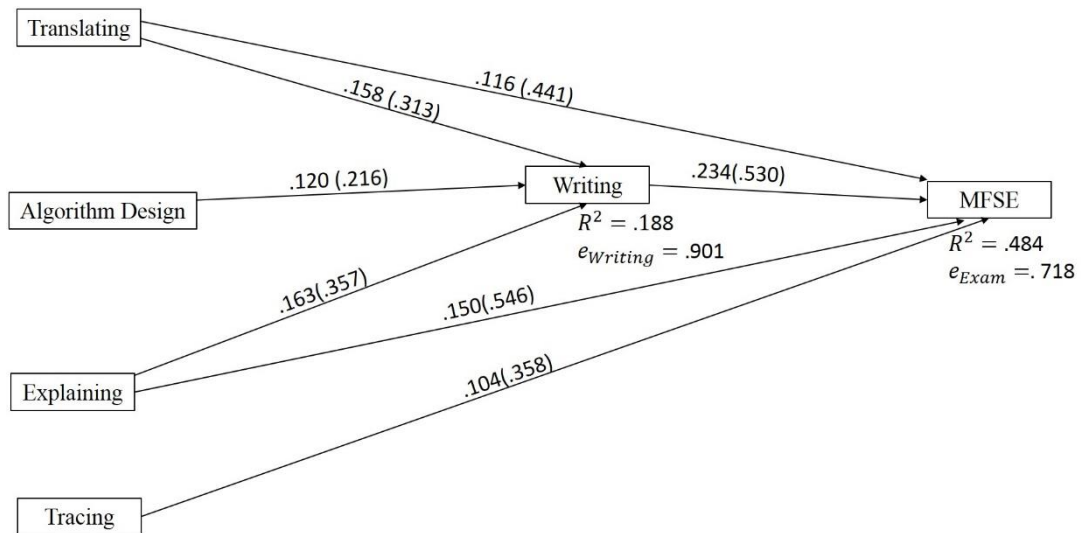


Figure 6.4. Path diagram with MFSE and writing as dependent variables and programming skills as independent variables.

From the path diagrams, we can observe that the results using MFSE as a dependent variable explain more of the variance in the model than do FSE and OS. The following hierarchy was reported after examination of the multiple regression results and path diagram. Number 1 (algorithm design) contributed least to student performance and number 5 (writing) contributed most to student performance:

1. algorithm design
2. tracing
3. translating
4. explaining
5. writing.

## 6.4 Summary

This chapter presented results on the quantitative bivariate and multivariate analysis of the student PST and survey data. From the correlation analysis, writing and explaining were shown to have a greater degree of association to student performance,

followed by translating and tracing, with algorithm design having the least association. This was consistent with the deep approach of learning having positive correlation to student performance, and surface approach of learning having a negative correlation to student performance. Although correlation coefficients were consistent, the values were either very low or not significant, mainly due to the fact that the English is not the students'/participants' first language. This may have made it difficult for them to comprehend the questions.

From the regression analysis, several direct and indirect programming skills were identified as predictors of student performance (FSM, OS and MFSE). In each of the students' performance taken as a dependent variable, algorithm design was found to be an indirect predictor via writing, while translating, explaining, tracing and writing were found to be direct predictors of student performance.

From path analysis, multiple regression results using MFSE as a dependent variable explained more of the variance in the model (48.4 percent) compared to using FSE (45 percent) and OS (36.5 percent) as dependent variables.

After examination of the multiple regression results and path diagram, the hierarchy of programming skills in terms of their contribution to student performance was reported in this order: algorithm design (least contribution), tracing, translating, explaining and writing (greatest contribution) to student performance.

The results presented in this chapter along with the results presented in Chapter 4 and 5 will be discussed further in Chapter 7.

## **Chapter 7: Answers to Research Questions and Discussion**

### **7.1 Introduction**

Chapter 6 presented the results of the bivariate and multivariate analysis of the quantitative data. This chapter discusses the results in light of the research questions. Research questions will be first answered and then discussed. The results of this study will be compared and contrasted with those from previous studies where possible. The key variables identified in the qualitative data will also be discussed. Recommendations for CS1 at RUB will be made, followed by a summary of this chapter.

### **7.2 Answers to Research Questions and Discussion**

#### **7.2.1 What was the students' prior computing experience and does this affect performance in CS1?**

##### ***7.2.1.1 Prior non-programming computer experience***

Students' prior non-programming computer experience was mostly in Internet searching, computer games and application software such as Microsoft Word, PowerPoint and Excel respectively. The statistical analysis did not show any significant result between students who spent more hours on information searches and application software and their performance in CS1. The only significant result was a negative correlation between students' spending more hours on computer games and their performance in CS1.

The results from this study showed no significant difference between students with prior non-programming computer experience and students without prior non-programming computer experience to student performance in CS1. The only significant correlation found was between students' spending more hours on computer



games (negative). There was little research on non-programming computer experience and its impact in students' performance in CS1. Studies that have examined this also reported that the prior non-programming experience does not affect student performance in CS1, except computer gaming experience, which has a negative impact on students' performance in CS1. Five of the eight lecturers interviewed also said that this would not affect students' performance in CS1. However, it would be easier for students initially in CS1 classes to have at least basic computer knowledge on how to use and understand what computers can do (i.e., word processing and spreadsheets).

While students were not exposed to non-programming computer activities in Y12 (especially how to Google and search for information and basic knowledge of Office), all student were exposed to non-programming computer activities during the time they learnt CS1. For example, Sherubtse College (SC) curriculum has a module titled 'Office Productivity Tools' offered in the first semester to all new students. In this module, students learn how to create documents, manipulate and analyse data using spreadsheets, prepare presentations and develop simple databases. However, the other two participating colleges (College of Science and Technology (CST) and Jigme Namgyel Engineering College (JNEC)) do not offer this module in their curricula. Moreover, there is a university-wide module titled 'Academic Skills' offered to all first semester students. In this module, student learn academic-related skills such as academic writing, oral presentation, critical thinking and effective communication skills. Therefore, students with no prior non-programming computer experience were learning in parallel with CS1. Further research may be required to investigate the impact of learning non-programming computer activities in parallel with CS1 on students' performance in CS1.

### ***7.2.1.2 Prior programming experience***

Students' prior programming experience was mostly in C, Java, JavaScript and Python respectively. The statistical analysis showed that students with some programming experience performed better in CS1 than those without. The interview results indicated that students with experience in computer programming language found it easier to understand the concepts in CS1 classes than their friends with no experience in any programming language.

The results from this study—that prior programming experience is positively correlated with student performance—are consistent with the studies reviewed in Section 2.4.1. The researcher believes that having some programming knowledge before commencing CS1 would be advantage for students in the initial stage only, as the researcher has observed during the teaching of CS1 that students with no prior knowledge of programming find it difficult initially but later excel over students with prior programming knowledge. This aligns with the results of Wilcox and Lionelle (2018), who found that students with prior exposure to programming performed significantly better in CS1, but this reduces in the subsequent CS2 course. This is further discussed in Section 7.3.

## **7.2.2 What are students' and lecturers' experience/perceptions of first programming language, programming paradigm, programming environment and teaching/learning methods and practices?**

### ***7.2.2.1 First programming language, programming paradigm and programming environment***

- Lecturers and students stated that C should be the first programming language (see Table 4.10, 4.4.2.4, Table 5.5 and 5.3.2.5).

- The lecturer stated that the procedural programming paradigm should be taught for the beginners (see Sections 5.2.3 and 5.3.2.3).
- Both lecturers and students stated that Turbo C++ and Microsoft Visual Studio should be used as a programming environment to learn programming. Some students also mentioned PyCharm and Code::Blocks (see Sections 4.3.2.3, Table 5.4 and 5.3.2.4).

The first programming languages widely used to teach in the universities worldwide (as discussed in the literature) were Java, C, C++ and Python. C was chosen by most students and lecturers at RUB as the first programming language. The reason may be that C was taught and learnt as the first programming language over a decade ago and lecturers were adept in teaching C. Also, since lecturers were familiar with C, they were resistant to change to another programming language. Some lecturers stated that many popular programming languages are based on C, so if students learn C, they will not face any problems switching to another programming language. Some lecturers and students suggested starting with Python, as they heard from others that it was simple to grasp. Currently at RUB, computer science and information technology students study C in first semester, C++ in second semester and Java in the fourth semester. Students enrolled in engineering programs such as architecture, electrical, and electronics and communication study C in first semester and this is the only module of programming that they study during their program. Students enrolled in civil engineering study C in the first semester and C++ in the second semester only.

After the literature review, the researcher recommended either C, C++, Python or Java for use as the first programming language. This is because several research has reported that the choice of the first programming language does not have a deep impact on difficulties that students may experience in learning to program (Ivanović,

Budimac, Radovanović, & Savić, 2015; Xinogalos, Pitner, Ivanović, & Savić, 2018). What matters most is the overall quality of the course and the lecturer who delivers the instruction. Students stated that it would benefit them if experienced lecturers were assigned to teach CS1. This is true, as it is important for students studying CS1 to build good foundations for their subsequent education in programming.

At RUB, most lecturers identified the procedural paradigm as a suitable paradigm to introduce for beginners. However, some studies (Kölling, 1999) reported that it was not necessary to start with procedural paradigm; object-oriented paradigm could be introduced right away. At RUB, since C is used in CS1, it is a procedural programming paradigm. According to Aleksić and Ivanović (2016), most courses are based on procedural than object-oriented paradigms. Thus, after the review of the literature and based on the results of this study, the researcher believes that at RUB, the procedural paradigm should be taught in the first semester and the object-oriented paradigm should be introduced in later semesters using the same or different programming language for computer science and information technology students. Engineering students can use either procedural or object-oriented paradigms, as they study only in first semester (except civil students, who study in the second semester as well). Although C programming language was used at RUB in CS1, staff professional development might assist in trying other commonly used programming languages in CS1.

At RUB, lecturers and students chose Turbo C++ and Microsoft Visual Studio as programming environments to learn programming. Again, these programming environments have been used for more than a decade, so lecturers and students are familiar with it. The researcher has used Turbo C++ and Dev-C++ as a programming environment while teaching CS1. Although both Turbo C++ and Dev-C++ provide a

user-friendly interface, display clear and accurate error messages, allow easy typing, compilation and running, the researcher prefers Dev-C++, as it is convenient to use. Turbo C++ takes control of the screen and you cannot move the cursor out of the environment. Sometimes you cannot even maximise or minimise the screen. Most often, the program ends abruptly and must be reopened. Moreover, based on the experience of Turbo C++ interface, the researcher would argue that Turbo C++ is now obsolete and it is time to move into new programming environments that are both easy and convenient for beginners to use. Thus, it is worth considering other programming environment possibilities and trialling these before changing curricula.

#### ***7.2.2.2 Teaching/learning methods and practices***

For teaching/learning methods and practices, both lecturers and students stated that students benefit in learning CS1 from coding live in the class and programming in pairs or groups (see Table 4.11, Section 4.4.2.5, Table 5.6 and Section 5.3.2.6). A lecturer also stated that students should practise writing code independently (see Section 5.2.6). Students stated that reading materials online and watching YouTube tutorials helped them in learning to program (see Table 4.11).

Although the literature reported numerous teaching/learning methods and practices that might assist students in overcoming the difficulties of learning to program, at RUB, the teaching/learning methods and practices that they found beneficial for students were: live coding by lecturers in the lecture session and pair/group programming in the laboratory. Students stated that live coding by the lecturers has helped them understand the program better and made it easier to find errors. Similarly, working in pairs/groups has helped them debug programs and understand concepts better by exchanging knowledge and ideas among peers. Thus, based on these results, the researcher considers in continuing with these methods and

practices, as the benefits have also been reported in previous studies (Paxton, 2002; Rubin, 2013). In addition, the researchers considers it might help students who are slow in following the class by watching pre-recorded lecturers in their own time as this method has generated positive feedback from students in previous studies (Mohorovičić & Strčić, 2011).

In addition to learning in pair/groups, encouraging students to write code on their own without help from their pair/groups might also help them learn to program. This is important, as they learn to write programs independently. It may also be useful to guide students in how to search for relevant materials online that might also assist them in understanding the concepts better. Thus, teaching/learning methods and practices such as live coding, pair/group programming, independent coding, reading materials online and watching YouTube tutorials that have helped students in RUB to learn to program might assist other students taking CS1 in other universities worldwide. It may also inform lecturers' teaching methods and practices.

### **7.2.3 What is the association, if any, between students' performance in CS1 and students' Y12 performance in mathematics, physics and chemistry?**

The Pearson correlational analysis showed a low positive significant correlation between mathematics, physics and chemistry, and student performance (see Table 6.3).

Though correlations were found to be significant, these were low. Prior performance in mathematics had a slightly higher correlation with student performance in CS1 than did physics and chemistry. This may be true, as previous studies have also revealed that mathematics is a predictor of students' success in CS1 (Ayub & Karnalim, 2017; Bennedsen & Caspersen, 2005; Bergin & Reilly, 2005b;

Qian & Lehman, 2016; Wilson & Shrock, 2001). In contrast, Watson, Li, and Godwin (2014) reported no significant correlation between either mathematics, physics or chemistry, and student performance. Few studies have examined the association between students' Y12 performance in physics and chemistry and their performance in CS1.

At RUB, students in computer science and engineering programs are selected based on their Y12 scores in mathematics, physics and chemistry. Students must pass (by over 50 percent) in mathematics, physics and chemistry. Selections are based on merit ranking of computed value by the subject ability rating. The ability rating of mathematics and physics is 5 and chemistry is 4. The results from this study indicate that the performance of students' in Y12 mathematics, physics and chemistry have a significant impact on student taking CS1 classes. The researcher believes students who are competent in mathematics and scored highly in their Y12 examination might perform well in CS1, as both mathematics and programming problems require logic. Thus, the researcher proposes to some amendments in the ability ratings for the selection of students in a program that consists of a programming module in the first semester. For example, the ability rating for mathematics could be 5, physics could be 4 and chemistry could be 3. This ability rating may apply only to computer science and information technology students.

#### **7.2.4 What is the association, if any, between students' performance in CS1 and students' learning approach?**

The results shows consistent expected results, with a deep approach having a positive significant correlation to student performance, and a surface approach having a negative significant correlation to student performance. However, these correlations were very low (see Table 6.6). A possible explanation for this may be the Bhutanese

students' inability to fully comprehend the questionnaire items, as English is the second language in Bhutan. Furthermore during interviews, despite the low correlations, the students' who gave a description of using a deep approach to learning in CS1 did well as compared to those students who described using a surface approach to learning. Thus, it is likely that a deep approach to learning improves students' performance in CS1.

Despite the low correlations, the results were similar to previous studies—a positive trend between a deep approach and student performance, and a negative trend between a surface approach and student performance.

Examining the individual items of the Biggs questionnaire, some items under deep approach correlated positively to student performance and some surface approach elements correlated negatively to student performance. Elements under deep motive—such as 'I work hard at my studies because I find the material interesting'—and elements under deep strategy—such as 'I find most new topics interesting and often spend extra time trying to obtain more information about them' and 'I test myself on important topics until I understand them completely'—correlated positively to student performance. Elements under surface motive—'I do not find my course very interesting so I keep my work to the minimum', 'I find I can get by in most assessments by memorising key sections rather than trying to understand them' and 'I see no point in learning material which is not likely to be in the examination'—and elements under surface strategy—'I learn some things by rote, going over and over them until I know them by heart even if I do not understand them' and 'I believe that lecturers shouldn't expect students to spend significant amounts of time studying material everyone knows won't be examined'—correlated negatively to student performance.



Similarly, lecturers' response to an interview question on their insights into how students in CS1 should approach their learning to achieve success mostly fell under the deep approach of learning (see Section 5.3.2.10). Thus, the results indicate that students' approaches to learning is a factor that determines success in CS1. The lecturer might be able to encourage students taking CS1 to adopt deep approaches to learning, as learning approach can be changed based on context, even though the student may have their own preference of learning approach. One way to attend to this may be to allocate the first week of CS1 classes to orienting students on the learning strategies that may help them to achieve good performance in CS1 instead of delving straight into CS1 content.

#### **7.2.5 What is the association, if any, among the programming skill variables?**

The results showed a positive significant correlation ( $p < 0.01$ ) between students' ability in algorithm design, translating, tracing, explaining and writing (see Table 6.4).

According to BRACElet studies (see Section 2.4.7), students who did well on explaining tasks usually performed well on writing tasks, and students who could only trace code below 50 percent could not usually explain code. The results from this study support the results of the BRACElet studies, as some associations between the programming skills exist. The positive significant association between explaining and writing indicates that when students become competent in explaining written code, they can then be able to write code and vice versa. Similarly, the positive significant association between tracing and explaining, and between tracing and writing indicates that students who have mastered tracing skills can perform well in explaining and writing. Further, it appears that students who can translate the logic from algorithm

design into any high-level programming language could explain the translated program and ultimately write the complete executable program. Thus, the association among the programming skills indicates that these skills develop simultaneously and are not independent.

#### **7.2.6 What is the association, if any, between students' performance in CS1 and students' ability in programming skills?**

The results showed a positive significant association ( $p < 0.01$ ) between student performance and ability in algorithm design, translating, tracing, explaining and writing. Explaining had the highest correlation with student performance, followed by writing, translating, tracing and algorithm design (see Table 6.5).

The association indicates that both explaining and writing skills are equally important to student performance in CS1. For example, students who can explain the algorithm design correctly can translate this into a programming language and explain the translated program. Explaining skills can also apply to the program that has already been given, when the student has to go through the program and explain in plain English the purpose of the code. Thus, students who are competent in explaining across the programming skills are likely to do well in writing, and ultimately, the overall exam.

The association between algorithm design and student performance (which was the lowest of all programming skills) indicates that students who can do well only in algorithm design may not be able to perform well in CS1. Similarly, the association between translating and student performance, and tracing and student performance indicates that students' competencies in these skills are not sufficient to score well in CS1. Further, students need to acquire all five programming skills to perform well in CS1, as the examination in CS1 includes questions across all programming skills.

Thus, from these correlation results, it is reasonable to assume that these programming skill variables play a role in student performance in CS1.

### **7.2.7 Is there a hierarchy among students' programming skills in terms of their contribution to student performance in CS1?**

The following hypothesised hierarchy was reported after examination of the correlation and path diagram created in chapter 6 (see Section 6.3.2). Number 1 (algorithm design) contributed least to student performance in CS1 and number 5 (writing) contributed most to student performance in CS1. They are numbered in this order, as the researcher believes that this is the order in which programming skills should be taught to benefit students' learning:

1. algorithm design
2. tracing
3. translating
4. explaining
5. writing.

The hypothesised hierarchy reported in this study seems logical as per the researcher's experience teaching CS1. For example, teaching algorithm design first before moving on to other programming skills. It seems logical that students should first be taught algorithm design in parallel with tracing and explaining the algorithm design. Once student becomes competent in algorithm design and can explain and trace the algorithm, the process of translating should be taught. At this stage, students must be introduced to the features, syntax and grammars of the programming language. Once students becomes competent in translating, tracing and explaining the code should be taught. Lastly, students should be taught how to write programs.

Although the programming skills are listed in order, the researcher is not implying that the ranking after algorithm design should be developed in strict hierarchy as shown above. The researcher supports the idea that tracing and explaining overlap with algorithm design, translating overlaps with writing, and tracing and explaining overlap with writing. The programming skills after algorithm design should develop in parallel and reinforce each other. This is supported by the correlations between the skills.

The hierarchy reported in this study is also similar to the hierarchy reported in the literature (Lopez et al., 2008). According to Lopez et al. (2008), writing is at the highest level and tracing and explaining are intermediate levels. This study has added algorithm design and translating to the list of tracing, explaining and writing, which previous studies have not done as a whole. The multiple regression analysis showed that translating, tracing, explaining and writing skills directly contribute to student performance, while algorithm design indirectly contributes to student performance via writing. It is true that algorithm design may not contribute directly to student performance, as students have to learn translating, tracing, explaining and writing to be able to perform well in CS1.

Based on the results of this study, the researcher proposed the outline of CS1 over 15 weeks as shown in Table 7.1. At RUB, one semester runs for 18 weeks; three weeks of the 18 were removed for the mid-semester exam, FSE and practical exam (refer Appendix K for a detailed proposed module description). In one week, three hours of lecture classes and a one-hour tutorial were considered in addition to three hours of laboratory practical sessions in a week.

Table 7.1

*Researcher Proposed Outline of CS1*

Week	Topics and programming skills
1	Learning strategies
2–4	Algorithms and problem-solving (tracing and explaining an algorithm)
5	Introduce language features: data types, assignments, variable declaration, operators, expressions and simple input/output functions
6	Translate simple algorithms into programming code (translating). Using programming environment, write translated programs, compile, run and debug
7–10	Introduce language features: control structures, functions, arrays and files
11–12	Translate complex algorithms into programming code (translating). Manually execute the translated programs (tracing) and explain the purpose of the translated code (explaining)
13	Introduce simple code writing from the given problem (writing) using programming environment. Manual tracing and explaining should be parallel as well
14	Introduce complex code writing from the given problem (writing) using programming environment. Manual tracing and explaining should be parallel as well
15	Introduce additional language features depending on the programming language used in CS1 involving tracing, explaining and writing programs.

**7.3 Suggestions on how to Improve Teaching/Learning of CS1 at RUB**

In the student and lecturer qualitative data on how to improve teaching/learning in CS1 at RUB, the most frequently mentioned areas by both students and lecturers were: conduct theory and practical classes in parallel, CS1 to be taught by an

experienced lecturer, encourage pair/group programming, live coding, offer the same tutor for both theory and practical sessions and prior programming experience.

**a) Conduct theory and practical classes in parallel**

Although most students preferred to have both the theory and the practical classes running in parallel in the computer laboratory, the researcher believes (based on their prior teaching experience at RUB) that this is not feasible, since there are a limited number of computer laboratories that cater to all students enrolled in a program. Moreover, the computer laboratory can accommodate only up to 35 students at a time. There are over 100 students enrolled in civil engineering. As an alternative, as one student suggested that lecturers should allow students to bring their laptops to the theory class to enable them to write programs when the lecturer is live coding in the class. Currently, we do not have that practice at SC and CST. Participants from JNEC were taught theory and practical classes at the same time in the computer laboratory, since it was the only program (Diploma in Computer Hardware and Networking) with computer laboratory components during the time of data collection. It may no longer be feasible, as JNEC has introduced degree programs in engineering that may require computer laboratories.

**b) CS1 to be taught by an experienced lecturer**

Both lecturer and students recommended that experienced lecturers be assigned to teach CS1. Most participants in this study were taught by four lecturers, who had just graduated from their degree programs. Only some participants were taught by the remaining four lecturers, who had completed their master's degree and have 1–2 years' experience teaching CS1. The researcher strongly supports assigning an experienced lecturer to teach CS1, as CS1 is the foundation module and building strong foundation is a necessity for the success of the remaining computer subjects in latter semesters.

**c) Encourage pair/group programming and live coding**

Both students and lecturers proposed including pair/group programming in the teaching/learning of CS1. Previous studies (see Chapter 2) on practising pair/group programming reported that this was effective for students in learning CS1. Similarly, live coding by lecturers was reported in previous studies to be beneficial for students, as students can observe the lecturer type the programs from scratch, and in the process, explaining and encouraging students to participate. In this way, students can also learn to compile, execute and debug programs. Lecturers can also give small programs to write in groups and let one student from each group code live in front of their peers. This might allow students to gain confidence in writing programs and motivate them.

**d) Same tutor for both theory and practical sessions**

Students expressed a preference to have only one tutor teaching both theory and practical classes. This does not happen all the time at RUB. However, sometimes when the theory tutor was given other modules to teach due to a shortage of tutors in that module, practical sessions were given to other tutors with lighter teaching loads. As long as the theory and practical tutor coordinate with each other this should not be a problem.

**e) Prior programming experience**

Some lecturers and students said that students should be exposed to programming concepts in Y12 so they do not feel overwhelmed when they are introduced to CS1. Previous research and this study has shown that students with prior programming experience perform better in CS1. Conversely, other studies showed prior programming experience does not have any impact on students' performance in CS1. While some studies reported on students' success in CS1 with and without prior programming experience, the specific programming experience that students should

have when learning CS1 was not specified. Thus, further research is required to explore which programming-related concepts students should learn in Y12 that may benefit them in learning CS1. For example, computational thinking and problem-solving or programming concepts using a programming language.

## **7.4 Recommendations for CS1 at RUB**

This thesis set out to investigate the factors that may affect the performance of students' studying CS1 at RUB. The following recommendations are made for CS1 at RUB based on the findings of this study:

### **1. Students' prior computing experience**

The researcher recommends that RUB collaborate with the School Education and Curriculum Division, of the Ministry of Education, Bhutan to explore avenues to enhance the existing curriculum by incorporating a subject in Y12 for all the science students that covers basic computer knowledge on Office packages, basic programming concepts and problem-solving skills. Moreover, the module on 'Office Productivity Tools' could then be removed from the first semester.

### **2. First programming language, programming paradigm, programming environment and teaching/learning methods and practices**

The researcher recommends RUB retain procedural paradigm with either C, C++, Java or Python as the programming language in the first semester and introduce object-oriented paradigm in later semesters using the same or different programming language for computer science and information technology students. Although C programming language is used at RUB in CS1, staff professional development might assist in trying other commonly used programming languages in CS1, such as Java and



Python. In regard to programming environment, the researcher highly recommends consideration of a simple and current environment as an alternative to Turbo C++ (obsolete) and Microsoft Visual Studio (sophisticated).

For teaching/learning methods and practices in CS1, the researcher highly recommends that lecturers allow students to bring their laptops to theory classes so they can write programs when the lecturer is live coding in class. Thus, students can obtain hands-on practice immediately after the theory sessions or during live coding by the lecturers. Also, students should be encouraged to work in pairs/groups initially, and later encouraged to practise writing code independently. Also, lectures should be recorded and uploaded to the course webpage for use by slow learners and students who wish to revise.

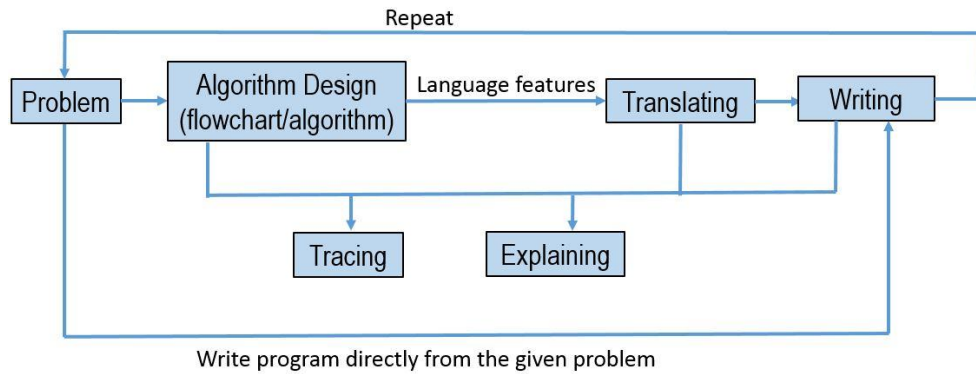
### **3. Students' learning approach**

The researcher recommends that lecturers at RUB encourage students to adopt deep approaches of learning before starting CS1 classes. One week of CS1 classes may be allocated to teach learning strategies that students can adopt to achieve success in CS1.

### **4. Teaching/learning approach of programming skills in CS1**

The researcher recommends the approach of teaching/learning programming skills in CS1 as shown in Figure 7.1. Begin with the introduction of how to solve simple problems using algorithm design (flowchart and algorithm). Then introduce simple language features like variables, data types, simple input/output functions, assignments and operators. This should be followed by translating the logic from algorithm design into programming language. Then, start writing the translated programs using the programming environment, compile, debug and execute the

program. Demonstrate manual tracing of the written program and show how to summarise the purpose of the written program (explaining). Students should also be taught how to trace and explain algorithm design and translated programs.



*Figure 7.1.* Approach of teaching/learning in CS1.

The process of algorithm design, translating, tracing, explaining and writing should be repeated to solve problems with increased difficulty level. New programming language features should be introduced, such as arrays, strings, user defined functions and input/output functions of arrays and strings. As students become competent in algorithm design, translating, tracing and explaining, they can be taught how to write programs directly once given the problem, with or without translating, depending on the students' capabilities.

## 7.5 Summary

This chapter has discussed the results from this study and answered the research questions. The first research question, which addressed students' prior computing experience and its impact on student performance in CS1, was discussed.

The second research question, which focused on students' and lecturers' experience/perceptions of first programming language, programming paradigm, programming environment and teaching/learning methods and practices, was discussed. The results provided information relating to first programming language

and paradigm to be taught in CS1, the programming environment to be adapted in learning to program and teaching/learning methods and practices that might assist students in learning to program and improve student performance in CS1.

The third research question focused on students' Y12 performance in mathematics, physics and chemistry and its association to student performance in CS1. The results of this question were discussed. The fourth research question—students' learning approaches and their impact on student performance in CS1—was discussed. The fifth research question, which focused on the relationship among the programming skill variables, was discussed. The sixth research question which focused on students' ability in programming skills and their impact on student performance in CS1 was discussed.

The seventh research question was whether there is hierarchy among the programming skills in terms of their contribution to student performance in CS1. The possible hierarchy presented was: 1) algorithm design, 2) tracing, 3) translating, 4) explaining, and 5) writing, where algorithm design was the lowest level and writing was the highest level. However, associations among the programming skills indicated that skills develop in parallel and reinforce each other.

This chapter also discussed the key variables identified from the qualitative data. This was followed by the recommendations for CS1 at RUB.

# Chapter 8: Conclusion

## 8.1 Introduction

Chapter 7 answered the research questions and discussed the results. This chapter provides a brief description of the preceding chapters and addresses each research question. Wider implications and contributions, limitations and directions for future research are indicated, and concluding remarks are provided.

Chapter 1 introduced this study and presented the origin of this thesis, provided background information, outlined research objectives and presented an overview of the thesis. Chapter 2 established the theoretical framework and reviewed the literature that examined the key variables under investigation in this study. Chapter 3 presented the detailed description of the methodology used for this study. Chapter 4 and 5 presented descriptive statistics and qualitative results of the data. Chapter 6 presented bivariate and multivariate analyses of the data. Chapter 7 answered the research questions and discussed the findings.

## 8.2 Research Findings

This section briefly summaries the results reported in Chapters 4, 5 and 6. Students' prior non-programming computer experience, such as Internet information searches and use of Office tools does not have any significant association with students' performance in CS1, except computer games, which had a negative significant association with student performance. Students with some programming skills in C, Java, JavaScript and Python performed better in CS1 and the association was significant and positive.

Both lecturers and students perceived that C should be the first programming language for CS1 and; Turbo C++ and Microsoft Visual Studio should be used as

programming environment to learn C. Moreover, both lecturers and students stated it would assist students in learning to program by practising teaching/learning methods such as live coding by the lecturer in class, pair/group programming, independent coding, reading materials online and watching YouTube tutorials. This study also identified a positive significant correlation between student performance in CS1 and students' Y12 performance in mathematics, physics and chemistry.

Further, this study showed that the deep approach of learning had a positive significant correlation to student performance in CS1, while the surface approach of learning had a negative significant correlation to student performance in CS1.

In addition, the results showed a positive significant correlation among the programming skill variables: algorithm design, translating, tracing, explaining and writing. This indicates that these skills develop in parallel and reinforce each other.

Moreover, the results showed a positive significant correlation between student performance and ability in programming skills, which suggests a possible hierarchy in programming skills. The regression analysis further showed that algorithm design contributes to student performance via writing, and writing contributes most to student performance. Thus the hypothesised hierarchy was reported as algorithm design, tracing, translating, explaining and writing, where algorithm design is at the lowest level and writing is at the highest in the hierarchy.

### **8.3 Wider Implications and Contribution**

This study has contributed new findings in the areas that might affect student performance in CS1: prior computing experience, first programming paradigm and language, programming environment, teaching/learning methods and practices, Y12 performance in mathematics, physics and chemistry, students' learning approach, programming skills and new SOLO descriptions.

The results from prior computing experience provide information in regard to the requirement of prior non-programming computer experience and programming experience. The results from this study may benefit educators in selecting students to study CS1.

Additionally, the results from first programming paradigm and language, programming environment, teaching/learning methods and practices provide clear insight into the aspects of lecturers' and students' perceptions/experience at RUB. The practice of live coding and pair/group programming in learning to program confirms its benefits. Thus, the information from this study can be utilised by educators to inform curricula decisions.

Further, students' Y12 performance in mathematics, physics and chemistry also provides clear insight for educators in regard to the importance of these subjects prior to taking CS1 classes.

Moreover, the positive impact of students' deep approach of learning in CS1 on student performance in CS1 provides information to both students and lecturers to encourage a more productive approach to learning in CS1. In addition, the PST questions developed by the researcher to measure students' skills across the five programming skills (with algorithm design and translating skills added) contributes to the body of knowledge in CS1 programming skills. This will also provide a foundation for further research in terms of the new variables introduced that are important in teaching CS1 as well as drawing further attention to the current variables for existing computing tertiary educators.

Subsequently, the SOLO taxonomy adapted to evaluate students' responses to PST questions also contributes new information in regard to the SOLO descriptions for each programming skill, thereby providing avenues for comparison.

## 8.4 Limitations

The main goal of this study was to investigate the factors that may affect student performance in CS1 at RUB. Although this goal has been achieved, the study was limited in several ways.

This study involved students enrolled in science and engineering programs from three colleges that are geographically spread across the country. This posed several limitations. One such limitation was that one cohort of students enrolled in B.Sc. Physical Science could not be included in this study, as these students study CS1 in their second semester, while students enrolled in other programs identified for this research study CS1 in the first semester. As data were collected immediately after students' completion of CS1 in their first semester, there would be a gap of five months if this study were to include those students. It would not only be expensive to travel back to these students after five months, but also the reliability and the validity of the programming skills test (PST) instrument may no longer hold after such a long gap. Therefore, the researcher decided to exclude these students for practical and pragmatic reasons.

For the same geographic reasons, the researcher could not administer the PST personally, as it would be a challenge for the researcher to travel from one college to another due to rugged terrains in Bhutan. Thus, the PST had to be administered by lecturers who volunteered to assist the researcher in their respective colleges at the same time. Further, considering the time that students spent answering the PST questions, the number of questions under each category of programming skills was limited to two. Ideally, it could have been at least five, with varying difficulties, administering one or two programming skills at a time to obtain quality results.

Moreover, the PST and survey, individual and group interviews could not be administered at the same time due to the reasons mentioned above. The researcher travelled to colleges for survey and interviews only when the students joined colleges after six weeks of winter break. Although there was a gap of six weeks between the PST and the survey, this arrangement was made to secure maximum student participants, as students were eager to leave college after their last examination. The ideal situation would be for the PST, survey and interviews be conducted immediately after the students completed a semester course in CS1.

Another limitation was that the three participating colleges administered their own examinations, so the difficulty level of the content covered in these examinations may not be identical. That is, the FSE and OS performance used in this study to measure student performance may have been different in terms of content.

One possible limitation to this study could be that one college, Gyalpozhing College of Information Technology (GCIT), that offers a program that includes a module on CS1 under RUB could not be included in this study. This is because GCIT was officially inaugurated on 6 October 2017 and began enrolling its first cohort of students in July 2018. The data for this study were collected in November 2016 (PST) and February–March 2017 (survey and interviews).

Additionally, the experience and qualification of the lecturers teaching CS1 were not the same across the three colleges. One college had four lecturers who had just completed their degree and were searching for an opportunity to upgrade their qualification. We asked about their opinions and experiences in the survey and interviews regarding CS1; however, we were less likely to receive creative and innovative ideas, as they themselves did not have much experience and exposure.



Although the medium of instruction is in English, it is not the participants' first language, which may have affected their comprehension of the Bigg's survey questionnaire. However, English was not a problem for students in responding to the exam and PST questions used in this study as the students had gained familiarity with the language used throughout the course.

The final limitation was that the findings from this study were very localised and may not be generalisable to an international audience, as differences may exist between the educational system in Bhutan and other countries. However, the findings may contribute to the body of literature in computer science education.

## **8.5 Future Research Directions**

Future research directions could be in the following areas: confirmation, extension and further investigation of the path model.

One key element of this research was to extend the model, include other variables and further the statistical analysis. Further research could also be extended to investigate the specific type of programming-related experience that students should have in Y12 prior to the study of CS1.

Moreover, in the future, investigation of the path model could be conducted, with different groups of students from other countries & institutions to confirm or disconfirm the path model.

## **8.6 Summary and Concluding Remarks**

In summary, this thesis provides new information to the existing body of computing research. The study identified several factors that may influence student performance in CS1 at RUB: prior computing experience, first programming language,

programming paradigm, programming environment, teaching/learning methods and practices, Y12 performance in mathematics, physics and chemistry, student learning approaches and programming skills. Programming skills such as algorithm design and translating were extra variables added to the study and path model, which other literature has not yet reported.

Moreover, this is the first time a study in Bhutan has examined all the variables identified in this research, thereby contributing new information to the body of computer science education literature.

The results from this study have implications for lecturers teaching CS1 who are interested in assisting students in improving success in CS1, particularly at RUB in Bhutan.



## References

- Afriyie, B. S. (2007). *Introduction to Computer Fundamentals* (Second ed.). Victoria, BC Canada: Trafford Publishing.
- Aleksić, V. & Ivanović, M. (2016). Introductory programming subject in European higher education. *Informatics in Education, 15*(2), 163.
- Ayub, M. & Karnalim, O. (2017). Predicting outcomes in introductory programming using J48 classification. *World Transactions on Engineering and Technology Education, 15*(2), 132-136.
- Bennedsen, J. & Caspersen, M. E. (2005). An investigation of potential success factors for an introductory model-driven programming course. *Proceedings of the first International Workshop on Computing Education Research*, 155-163. doi:10.1145/1089786.1089801
- Bergin, S. & Reilly, R. (2005a). The influence of motivation and comfort-level on learning to program *Proceedings of the 17th Workshop of the Psychology of Programming Interest Group* (pp. 293-304). Brighton, UK: Psychology of Programming Interest Group.
- Bergin, S. & Reilly, R. (2005b, February 23 - 27). *Programming: Factors that influence success*. Paper presented at the 36th SIGCSE Technical Symposium on Computer Science Education St. Louis, Missouri, US. doi:10.1145/1047344.1047480
- Bevan, J., Werner, L. & McDowell, C. (2002). Guidelines for the use of pair programming in a freshman programming class. *Proceedings of the 15th Conference on Software Engineering Education and Training*, 100-107. doi: 10.1109/CSEE.2002.995202
- Bhattacharya, R. & Bhattacharya, B. (2015). Psychological factors affecting student's academic performance in higher education among students *International Journal for Research & Development in Technology, 4*(1), 63-71.
- Biggs, J., Kember, D. & Leung, D. Y. (2001). The revised two-factor study process questionnaire: R-SPQ-2F. *British Journal of Educational Psychology, 71*(1), 133-149. doi:10.1348/000709901158433
- Biggs, J. B. (1987). *Student approaches to learning and studying. Research monograph*. Melbourne: Australian Council for Educational Research.
- Biggs, J. B. & Collis, K. F. (1982). *Evaluating the quality of learning: The SOLO taxonomy (structure of the observed learning outcome)*. Melbourne, VIC: Academic Press.
- Biró, P., Csenoch, M., Abari, K. & Máth, J. (2016). First year students' algorithmic skills in tertiary computer science education (Kunifuji S., Papadopoulos G., Skulimowski A., Kacprzyk J. ed., Vol. 416, pp. 351-358). Berlin: Springer.
- Brilliant, S. S. & Wiseman, T. R. (1996). The first programming paradigm and language dilemma. *Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education, 28*(1), 338-342. doi:10.1145/236452.236572
- Buckley, C. A., Pitt, E., Norton, B. & Owens, T. (2010). Students' approaches to study, conceptions of learning and judgements about the value of networked technologies. *Active Learning in Higher Education, 11*(1), 55-65.
- Burton, P. J. & Bruhn, R. E. (2003). Teaching programming in the OOP era. *ACM SIGCSE Bulletin, 35*(2), 111-114. doi:10.1145/782941.782993

- Byrne, M., Flood, B. & Willis, P. (2002). Approaches to learning of European business students. *Journal of Further and Higher Education*, 26(1), 19-28. doi:10.1080/03098770120108275
- Byrne, P. & Lyons, G. (2001). The effect of student attributes on success in programming. *ACM SIGCSE Bulletin*, 33(3), 49-52.
- Campbell, P. F. & McCabe, G. P. (1984). Predicting the success of freshmen in a computer science major. *Communications of the ACM*, 27(11), 1108-1113.
- Chamillard, A. & Braun, K. A. (2000). Evaluating programming ability in an introductory computer science course. *ACM SIGCSE Bulletin*, 32(1), 212-216. doi:10.1145/331795.331857
- Clear, T., Whalley, J., Lister, R., Carbone, A., Hu, M., Sheard, J., et al. (2008). Reliably classifying novice programmer exam responses using the SOLO taxonomy. *National Advisory Committee on Computing Qualifications*.
- Cohen, L. (2017). *Research methods in education* (8th ed.): London: Taylor and Francis.
- Creswell, J. W. (2013). *Research design (International student edition): Qualitative, quantitative, and mixed methods approaches* (4th ed.). Thousand Oaks, US: Sage publications.
- Creswell, J. W. (2014). *Research design: Qualitative, quantitative, and mixed methods approaches* (4th ed.). London: Sage publications.
- Crews, T. & Ziegler, U. (1998). The flowchart interpreter for introductory programming courses. *Proceedings of the 28th Annual Frontiers in Education Conference. Moving from 'Teacher-Centered' to 'Learner-Centered' Education*, 307 - 312 doi: 10.1109/FIE.1998.736854
- de Raadt, M., Hamilton, M., Lister, R., Tutty, J., Baker, B., Box, I., et al. (2005). Approaches to learning in computer programming students and their effect on success. On *Proceedings of the 28th HERDSA Annual Conference: Higher Education in a Changing World*. Sydney, Australia Higher Education Research and Development Society of Australasia
- Decker, R. & Hirshfield, S. (1994). The top 10 reasons why object-oriented programming can't be taught in CS 1. *ACM SIGCSE Bulletin*, 26(1), 51-55.
- Dierbach, C. (2014). Python as a first programming language. *Journal of Computing Sciences in Colleges*, 29(6), 153-154.
- Freund, S. N. & Roberts, E. S. (1996). Thetis: An ANSI C programming environment designed for introductory use. *SIGCSE*, 96, 300-304. doi:10.1145/236452.236560
- Gadelrab, H. F. (2011). Factorial structure and predictive validity of approaches and study skills inventory for students (ASSIST) in Egypt: A confirmatory factor analysis approach. *Electronic Journal of Research in Educational Psychology*, 9(3), 1197-1218.
- Get started with Visual Studio 2017. Retrieved from <https://tutorials.visualstudio.com/vs-get-started/user-interface>
- Gloria, A. M. & Robinson Kurpius, S. E. (2001). Influences of self-beliefs, social support, and comfort in the university environment on the academic nonpersistence decisions of American Indian undergraduates. *Cultural Diversity and Ethnic Minority Psychology*, 7(1), 88. doi:10.1037/1099-9809.7.1.88
- Goel, A. (2010). *Computer fundamentals*: Pearson Education India.
- Gómez-Albarrán, M. (2005). The teaching and learning of programming: a survey of supporting software tools. *Computer Journal*, 48(2), 130-144. doi:10.1093/comjnl/bxh080
- Grover, P. S. (2001). *PASCAL Programming Fundamentals* (8 ed.): Allied Publishers Limited.

- Gupta, D. (2004). What is a good first programming language? *Crossroads*, 10(4), 7-7. doi:10.1145/1027313.1027320
- Hagan, D. & Markham, S. (2000). *Does it help to have some programming experience before beginning a computing degree program?* Paper presented at the ACM SIGCSE Bulletin.
- Hicks, A. (2010). Towards social gaming methods for improving game-based computer science education. *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, 259-261. doi:10.1145/1822348.1822386
- Hijaz, S. T. & Naqvi, S. R. (2006). Factors affecting students' performance: A case of private colleges in Bangladesh. *Journal of sociology*, 3(1), 44-45.
- Holden, E. & Weeden, E. (2003). The impact of prior experience in an information technology programming course sequence. *Proceedings of the 4th Conference on Information Technology Curriculum*, 41-46. doi:10.1145/947121.947131
- Hooshyar, D., Ahmad, R. B., Shamshirband, S., Yousefi, M. & Horng, S.-J. (2015). A flowchart-based programming environment for improving problem solving skills of CS minors in computer programming. *Asian International Journal of Life Sciences*, 24(2), 629-646.
- Ivanović, M., Budimac, Z., Radovanović, M. & Savić, M. (2015). Does the choice of the first programming language influence students' grades? *Proceedings of the 16th International Conference on Computer Systems and Technologies*, 305-312. doi:10.1145/2812428.2812448
- Jeyapoovan, T. (2015). *Fundamentals of Computing and Programming in C*: Vikas Publishing House Pvt. Ltd.
- Jimoyiannis, A. (2013). Using SOLO taxonomy to explore students' mental models of the programming variable and the assignment statement. *Themes in Science and Technology Education*, 4(2), 53-74.
- Kamtsios, S. & Karagiannopoulou, E. (2015). Exploring relationships between academic hardiness, academic stressors and achievement in university undergraduates. *Journal of Applied Educational and Policy Research*, 1(1).
- Kazimoglu, C., Kiernan, M., Bacon, L. & Mackinnon, L. (2012). A serious game for developing computational thinking and learning introductory computer programming. *Procedia-Social and Behavioral Sciences*, 47, 1991-1999.
- Kersteen, Z. A., Linn, M. C., Clancy, M. & Hardyck, C. (1988). Previous experience and the learning of computer programming: The computer helps those who help themselves. *Journal of educational computing research*, 4(3), 321-333.
- Kölling, M. (1999). The problem of teaching object-oriented programming, Part 1: Languages. *Journal of Object-oriented programming*, 11(8), 8-15.
- Kölling, M. & Rosenberg, J. (1996). An object-oriented program development environment for the first programming course. *Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education*, 28(1), 83-87. doi:10.1145/236452.236514
- Konvalina, J., Wileman, S. A. & Stephens, L. J. (1983). Math proficiency: A key to success for computer science students. *Communications of the ACM*, 26(5), 377-382.
- Krpan, D., Mladenović, S. & Rosić, M. (2015). Undergraduate programming courses, students' perception and success. *Procedia-Social and Behavioral Sciences*, 174, 3868-3872.
- Kruglyk, V. & Lvov, M. (2012, June 6 - 10). *Choosing the first educational programming language*. Paper presented at the International Conference on ICT in Education, Research, and Industrial

Applications:: Integration, Harmonization and Knowledge Transfer, Kherson State University, Kherson, Ukraine.

- Lipman, D. (2014). Learn CS1: A new, browser-based C programming environment for CS1. *Journal of Computing Sciences in Colleges*, 29(6), 144-150.
- Lister, R., Clear, T., Bouvier, D. J., Carter, P., Eckerdal, A., Jacková, J., et al. (2010). Naturally occurring data as research instrument: Analyzing examination responses to study the novice programmer. *ACM SIGCSE Bulletin*, 41(4), 156-173. doi:10.1145/1709424.1709460
- Lister, R., Fidge, C. & Teague, D. (2009). Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *Proceedings of the 14th annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education*, 41(3), 161-165. doi:10.1145/1562877.1562930
- Lopez, M., Whalley, J., Robbins, P. & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. *Proceedings of the Fourth international Workshop on Computing Education Research*, 101-112. doi:10.1145/1404520.1404531
- Malik, S. I. & Coldwell-Neilson, J. (2017). Comparison of tradition and ADRI based teaching approaches in an introductory programming course. *Journal of Information Technology Education: Research*, 16, 267-283.
- Mannila, L., Peltomäki, M. & Salakoski, T. (2006). What about a simple language? Analyzing the difficulties in learning to program. *Computer Science Education*, 16(3), 211-227. doi:10.1080/08993400600912384
- McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., et al. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, 33(4), 125-180.
- McDowell, C., Werner, L., Bullock, H. & Fernald, J. (2002). The effects of pair-programming on performance in an introductory programming course. *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, 34(1), 38-42. doi:10.1145/563517.563353
- Merrick, K. E. (2010). An empirical evaluation of puzzle-based learning as an interest approach for teaching introductory computer science. *IEEE Transactions on Education*, 53(4), 677-680. doi:10.1109/TE.2009.2039217
- Mohorovičić, S. & Strčić, V. (2011, September 21 - 23). *An overview of computer programming teaching methods*. Paper presented at the 22nd Central European Conference on Information and Intelligent Systems, Varaždin, Croatia.
- Mushtaq, I. & Khan, S. N. (2012). Factors affecting students' academic performance. *Global Journal of Management and Business Research*, 12(9).
- Patil, S. P. & Goje, A. C. (2009, April 17 - 20). *The effect of developments in student attributes on success in Programming of management students*. Paper presented at the 2009 International Conference on Education Technology and Computer, Singapore. doi: 10.1109/ICETC.2009.35
- Paxton, J. (2002). Live programming as a lecture technique. *Journal of Computing Sciences in Colleges*, 18(2), 51-56.
- Philpott, A., Robbins, P. & Whalley, J. (2007, July 8-11). *Assessing the steps on the road to relational thinking*. Paper presented at the 20th Annual Conference of the National Advisory Committee on Computing Qualifications, Nelson, New Zealand.
- Porter, L., Guzdial, M., McDowell, C. & Simon, B. (2013). Success in introductory programming: What works? *Communications of the ACM*, 56(8), 34-36. doi:10.1145/2492007.2492020

- Pritchard, M. E. & Wilson, G. S. (2003). Using emotional and social factors to predict student success. *Journal of College Student Development*, 44(1), 18-28. doi:10.1353/csd.2003.0008
- Qian, Y. & Lehman, J. D. (2016). Correlates of success in introductory programming: A study with middle school students. *Journal of Education and Learning*, 5(2), 73-83.
- Radenski, A. (2006). Python First: A lab-based digital introduction to computer science. *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, 38(3), 197-201. doi:10.1145/1140123.1140177
- Rubin, M. J. (2013). The effectiveness of live-coding to teach introductory programming. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, 651-656. doi:10.1145/2445196.2445388
- Sanders, I. D. & Langford, S. (2008). Students' perceptions of python as a first programming language at wits. *Proceedings of the 13th annual conference on Innovation and technology in computer science education*, 40(3), 365-365. doi:10.1145/1384271.1384407
- Schmeck, R. R., Geisler-Brenstein, E. & Cercy, S. P. (1991). Self-concept and learning: The revised inventory of learning processes. *Educational Psychology*, 11(3-4), 343-362. doi:10.1080/0144341910110310
- Schmider, E., Ziegler, M., Danay, E., Beyer, L. & Bühner, M. (2010). Is it really robust? Reinvestigating the robustness of ANOVA against violations of the normal distribution assumption. *Methodology: European Journal of Research Methods for the Behavioral and Social Sciences*, 6(4), 147 - 151. doi:10.1027/1614-2241/a000016
- Schoeman, M., Gelderblom, H. & Smith, E. (2012). A tutorial to teach tracing to first-year programming students. *Progressio—South African Journal for Open and Distance Learning Practice*, 34(3), 59-80.
- Shannon, C. (2003). Another breadth-first approach to CS I using python. *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, 35(1), 248-251. doi:10.1145/792548.611980
- Sheard, J., Carbone, A., Lister, R., Simon, B., Thompson, E. & Whalley, J. L. (2008). Going SOLO to assess novice programmers. *Proceedings of the 13th annual conference on Innovation and technology in computer science education*, 40(3), 209-213. doi:10.1145/1597849.1384328
- Shi, N., Cui, W., Zhang, P. & Sun, X. (2017). Evaluating the Effectiveness Roles of Variables in the Novice Programmers Learning. *Journal of educational computing research*, 0735633117707312.
- Shuhidan, S., Hamilton, M. & D'Souza, D. (2009). A taxonomic study of novice programming summative assessment. *Proceedings of the 11th Australasian Conference on Computing Education*, 95, 147-156.
- Sung, K., Hillyard, C., Angotti, R. L., Panitz, M. W., Goldstein, D. S. & Nordlinger, J. (2010). Game-themed programming assignment modules: A pathway for gradual integration of gaming context into existing introductory programming courses. *IEEE Transactions on Education*, 54(3), 416-427. doi:10.1109/TE.2010.2064315
- Tait, H. & Entwistle, N. (1996). Identifying students at risk through ineffective study strategies. *Higher education*, 31(1), 97-116.
- Tan, G. & Venables, A. (2010). Wearing the assessment 'BRACElet'. *Journal of Information Technology Education: Innovations in Practice*, 9(1), 25-34.



- Taylor, H. G. & Luegina, M. (1991). An analysis of success factors in college computer science: High school methodology is a key element. *Journal of Research on Computing in Education*, 24(2), 240-245.
- The Joint Task Force on Computing Curricula Association for Computing Machinery (ACM) IEEE Computer Society. (2013). *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. Retrieved from [https://www.acm.org/binaries/content/assets/education/cs2013\\_web\\_final.pdf](https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf)
- Thomas, L., Ratcliffe, M. & Robertson, A. (2003). Code warriors and code-a-phobes: a study in attitude and pair programming. *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, 35(1), 363-367. doi:10.1145/611892.612007
- Trigwell, K. & Prosser, M. (1991). Improving the quality of student learning: The influence of learning context and student approaches to learning on learning outcomes. *Higher education*, 22(3), 251-266.
- Tshering, P., Lhamo, D., Yu, L. & Berglund, A. (2017). How do first year students learn C programming in Bhutan? *International Conference on Learning and Teaching in Computing and Engineering*, 25-29. doi:10.1109/LaTiCE.2017.12
- Van Roy, P. (2009). Programming paradigms for dummies: What every programmer should know. *New computational paradigms for computer music*, 104, 616-621.
- Venables, A., Tan, G. & Lister, R. (2009, August 10 - 11). *A closer look at tracing, explaining and code writing skills in the novice programmer*. Paper presented at the Fifth International Workshop on Computing Education Research Workshop, Berkeley, CA, US. doi:10.1145/1584322.1584336
- Watson, C., Li, F. W. & Godwin, J. L. (2014). No tests required: Comparing traditional and dynamic predictors of programming success. New York, NY: Association for Computing Machinery
- Werth, L. H. (1986). Predicting student performance in a beginning computer science class. *ACM SIGCSE Bulletin - Proceedings of the 17th SIGCSE Technical Symposium on Computer Science Education*, 18(1), 138 - 143. doi:10.1145/953055.5701
- Whalley, J., Clear, T., Robbins, P. & Thompson, E. (2011). Salient elements in novice solutions to code writing problems. *Proceedings of the 13th Australasian Computing Education Conference*, 114, 37-46.
- Wiedenbeck, S., Labelle, D. & Kain, V. N. (2004, April 5-7). *Factors affecting course outcomes in introductory programming*. Paper presented at the 16th Annual Workshop of the Psychology of Programming Interest Group, Carlow, Ireland.
- Wiedenbeck, S., Ramalingam, V., Sarasamma, S. & Corritore, C. L. (1999). A comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interacting with Computers*, 11(3), 255-282. doi:10.1016/S0953-5438(98)00029-0
- Wilcox, C. & Lionelle, A. (2018). Quantifying the benefits of prior programming experience in an introductory computer science course. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 80-85. doi:10.1145/3159450.3159480
- Williams, L., Wiebe, E., Yang, K., Ferzli, M. & Miller, C. (2002). In support of pair programming in the introductory computer science course. *Computer Science Education*, 12(3), 197-212. doi:10.1076/csed.12.3.197.8618
- Wilson, B. C. (2002). A study of factors promoting success in computer science including gender differences. *Computer Science Education*, 12(1-2), 141-164. doi:10.1076/csed.12.1.141.8211

- Wilson, B. C. & Shrock, S. (2001). *Contributing to success in an introductory computer science course: A study of twelve factors*. Paper presented at the 32nd SIGCSE Technical Symposium on Computer Science Education, Charlotte, NC. doi:10.1145/364447.364581
- Winslow, L. E. (1996). Programming pedagogy—A psychological overview. *ACM SIGCSE Bulletin*, 28(3), 17-22. doi:10.1145/234867.234872
- Xinogalos, S., Pitner, T., Ivanović, M. & Savić, M. (2018). Students' perspective on the first programming language: C-like or Pascal-like languages? *Education and Information technologies*, 23(1), 287-302. doi:10.1007/s10639-017-9601-6
- Yadin, A. (2011). Reducing the dropout rate in an introductory programming course. *ACM inroads*, 2(4), 71-76. doi:10.1145/2038876.2038894
- Yang, H.-J. (2004). Factors affecting student burnout and academic achievement in multiple enrollment programs in Taiwan's technical-vocational colleges. *International Journal of Educational Development*, 24(3), 283-301. doi:10.1016/j.ijedudev.2003.12.001
- Yusoff, M. S. B., Rahim, A. F. A., Baba, A. A., Ismail, S. B. & Pa, M. N. M. (2013). Prevalence and associated factors of stress, anxiety and depression among prospective medical students. *Asian Journal of Psychiatry*, 6(2), 128-133. doi:10.1016/j.ajp.2012.09.012
- Zaffar, M., Hashmani, M. A. & Savita, K. (2018). *A Study of prediction models for students enrolled in programming subjects*. Paper presented at the 4th International Conference on Computer and Information Sciences Kuala Lumpur, Malaysia

Every reasonable effort has been made to acknowledge the owners of copyright material. I would be pleased to hear from any copyright owner who has been omitted or incorrectly acknowledged.



## **Appendices**



# Appendix A: Programming Skills Test Instrument



## CS1 TEST Instrument

*Please enter the details below immediately upon receipt of this paper.*

Programme: .....  
Student Number: .....  
Sex (F/M): .....  
Age: .....  
Date: .....

### Instructions to Student Participants

- ✓ You will get **1.5 Hrs** in total to write this TEST.
- ✓ ALL answers must be written in the answer sheet provided.
- ✓ ALL QUESTIONS ARE COMPULSORY.
- ✓ This TEST comprises of five sections with two questions each.
- ✓ You are strongly advised not to leave any blank spaces even if you don't know the answer. Instead explain the question in your own words on how you have understood.

## CSI TEST Instrument

### Section A: Algorithm Design (Flowchart/ Algorithm)

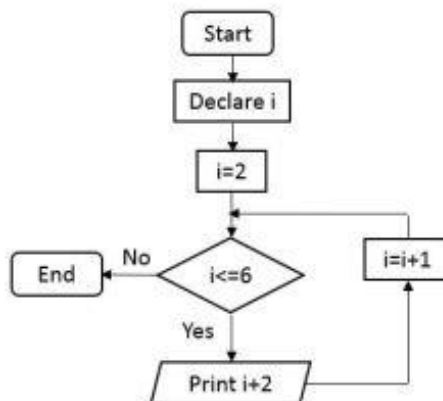
Write an algorithm or draw flowchart for the following questions:

1. Find the average of three numbers entered by the user and display the result.
2. Find the smallest number among the three numbers entered by the user and display the result.

### Section B: Translating

Translate the following algorithm and flowchart into C programming language.

1.
  1. Start
  2. Declare variable a, b, BIG, SMALL.
  3. Read a, b
  4. If a is less than b then
    - 4.1. BIG = b
    - 4.2. SMALL = a
  5. Else
    - 5.1. BIG = a
    - 5.2. SMALL = b
  6. Write BIG, SMALL
  7. End
- 2.



## CS1 TEST Instrument

### Section C: Tracing

Study the given piece of code and answer the question that follows:

1.

```

4 | int x;
5 | if(x<2)
6 |     printf("%d",x);
7 | else if(x<4)
8 |     printf("%d", 2*x);
9 |     else if(x<6)
10 |         printf("%d",3*x);
11 |         else
12 |             printf("%d", 4*x);
13 |

```

- Which branch of **if/else** structure will be executed if  $x = 5$ .
- What is the output of the program when  $x=6$ .

2.

```

4 | int i, j, x = 0;
5 | for (i = 0; i < 3; ++i)
6 | {
7 |     for (j = 0; j < i; ++j)
8 |     {
9 |         x += (i + j - 1);
10 |     }
11 | }
12 | printf("x = %d", x);

```

- Write the value of **x** when **i = 1**
- Write the value of **x** at **line 12**.

### Section D: Explaining

Explain in plain English the purpose of the following code:

1.

```

4 | int a =3;
5 | int b = 7;
6 | int c;
7 |
8 | c = a;
9 | a = b;
10 | b = c;
11 |
12 | printf("a = %d, b= %d ",a,b);

```



## CSI TEST Instrument

2.

```
int i,c=0;
int a[10]= {2,3,4,5,7,8,9,12,34,6};

for(i=0;i<10;i++)
{
    if(a[i]%2==0)
    {
        c++;
    }
}
printf("%d",c);
```

### Section E: Writing

1. Write a program in C that reads two integer number in the variables **x** and **y**. Divide **x** by **y** and report the result. You should confirm that **y** (divisor) is not zero before performing the division, as division by 0 is not possible and will cause your program to crash.
2. Write a program in C that calculates the sum of every third integer, beginning with **1 = 2** (i.e. calculate the sum of **2 + 5 + 8 + 11 + .....**) for all values of **1** that are less than 30.

***THANK YOU FOR YOUR PARTICIPATION***

## Appendix B: Programming Skills Test Marking Criteria Using SOLO

### Classification

#### Section A: Algorithm Design

Q1: Find the average of three numbers entered by the user and display the result.

A suitable algorithm and flowchart solution may look like this:

#### Algorithm

*Step 1: Start*

*Step 2: Declare variables a, b, c and avg*

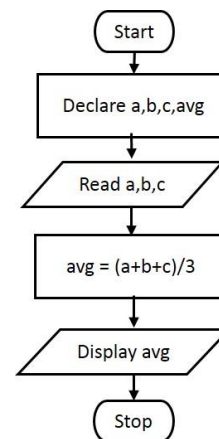
*Step 3: Read a, b and c.*

*Step 4: Compute  $avg = (a+b+c)/3$*

*Step 5: Display avg.*

*Step 6: Stop*

#### Flowchart



Q2: Find the smallest number among the three numbers entered by the user and display the result. A suitable algorithm and flowchart solution may look like as shown:

#### Algorithm

*Step 1: Start*

*Step 2: Declare variables a, b and c.*

*Step 3: Read a, b and c.*

*Step 4: If  $a < b$*

*If  $a < c$*

*Display a is the smallest number.*

*Else*

*Display c is the smallest number.*

*Else*

*If  $b < c$*

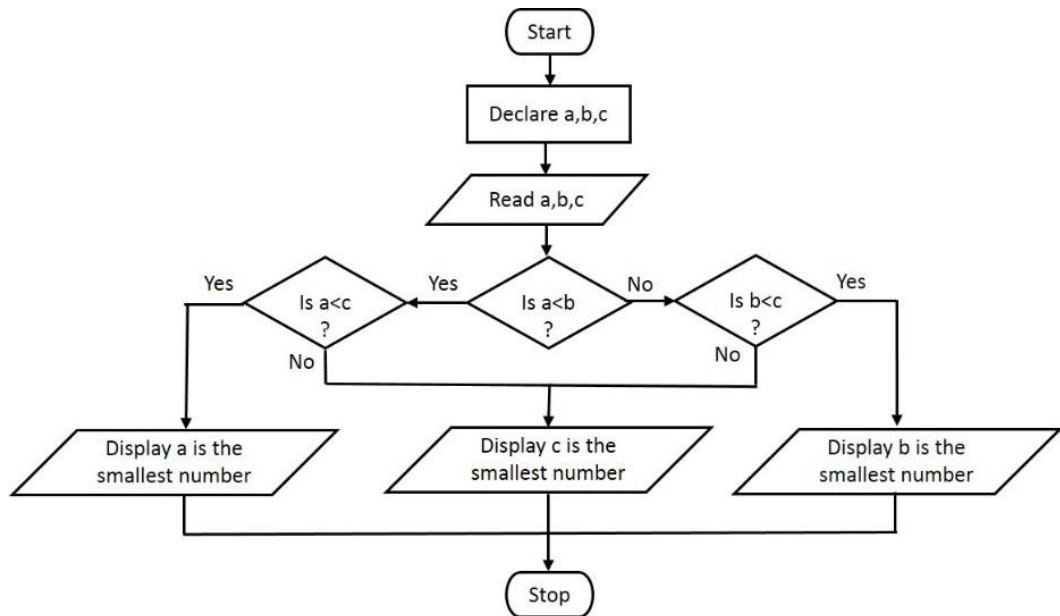
*Display b is the smallest number.*

*Else*

*Display c is the smallest number.*

*Step 5: Stop*

## Flowchart



The following components are examined in students' response to algorithm:

- Ability to declare variables (E-U).
- Ability to externally supply inputs (E-S).
- Ability to show the computation (E-S for Q1) and (H-S for Q2).
- Ability to show what output is produced (E-S).
- Ability to write clear and unambiguous instructions (E-U).
- Ability to terminate the algorithm in a finite number of steps (E-U).
- Ability to write algorithm in correct logical order (H-U).

The following components are examined in students' response to flowchart:

- a) Ability to declare variables (E-U).
- b) Ability to externally supply inputs (E-S).
- c) Ability to show the computation (E-S for Q1) and (H-S for Q2).
- d) Ability to show what output is produced (E-S).
- e) Ability to write clear and unambiguous instructions (E-U).
- f) Ability to terminate the flowchart in a finite number of steps (E-U).
- g) Ability to draw flowchart in correct logical order (H-U).
- h) Ability to draw flowchart with correct symbols (H-U).

Table B.1

*SOLO classification for Algorithm Design Questions*

SOLO level	Indicator	Raw Score
Relational [R]	Able to complete all of the components to form a coherent whole.	5
Relational Error [RE]	Able to complete all of the components but has some minor errors or omissions.	4
Multistructural [M]	Able to complete most of the components. All H-S and E-S components complete and valid.	3
Multistructural Error [ME]	Able to complete most of the components but has some minor errors or omissions.	2
Unistructural [U]	Able to complete some components only.	1
Prestructural [P]	There are pieces which makes no sense or the answer is totally wrong.	0
No attempt [N]	The answer is blank.	999

## Section B: Translating

Q1:

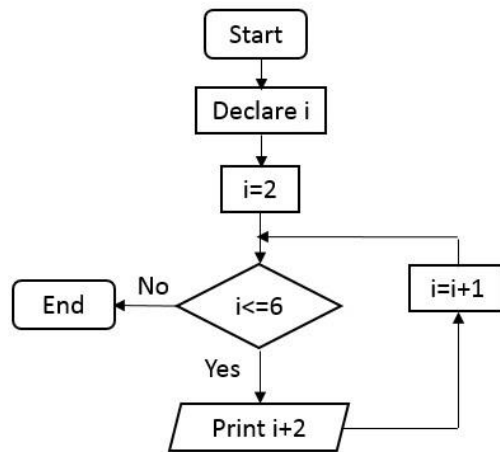
A suitable solution may look like this:

- |                                       |   |
|---------------------------------------|---|
| 1. Start                              |   |
| 2. Declare variable a, b, BIG, SMALL. |   |
| 3. Read a,b                           | 1. <i>int a,b,BIG,SMALL;</i>                                    |
| 4. If a is less than b then           | 2. <i>scanf("%d %d", &amp;a,&amp;b);</i>                        |
| 4.1. BIG = b                          | 3. <i>if(a&lt;b)</i>  |
| 4.2. SMALL = a                        | 4. <i>{</i>   |
| 5. Else                               | 5. <i>BIG = b;</i>  |
| 5.1. BIG = a                          | 6. <i>SMALL = a;</i>  |
| 5.2. SMALL = b                        | 7. <i>}</i>   |
| 6. Write BIG, SMALL                   | 8. <i>else {</i>  |
| 7. End                                | 9. <i>BIG=a;</i>  |
|                                       | 10. <i>SMALL = b; }</i>   |
|                                       | 11. <i>printf("Big = %d, Small = %d",</i><br><i>BIG,SMALL);</i> |

The following components are examined in students' response to Translating Q1:

- Ability to declare variables of correct data types -Line 1 (E-S).
- Ability to read the variables from the console -Line 2 (E-S).
- Ability to use **if/else** statement with correct computation - Line 3-10 (E-S).
- Ability to show the desired output - Line 11 (E-S).
- Ability to write well-structured program in clear logical order (E-U).

Q2:



A suitable solution may look like this:

```

1. int i;
2. for(i=2;i<=6;i++)
   {
       printf("%d", i+2);
   }
  
```

OR

```

1. int i=2;
2. while(i<=6)
3. {
4.     printf("%d", i+2);
5.     i=i+1;
6. }
  
```

The following components are examined in students' response to Translating Q2:

- a) Ability to declare variables of correct data types -Line 1 (E-U).
- b) Ability to formulate correct loop - Line 2 (H-S).
- c) Ability to show the desired output - Line 4 (E-S).
- d) Ability to write well-structured program in clear logical order (E-U).

Table xx shows the SOLO classification for Translating questions

Table B.2

*SOLO Classification for Translating Questions*

SOLO level	Indicator	Raw Score
Relational [R]	Able to complete all of the components to form a coherent whole.	3
Multistructural [M]	Able to complete most of the components. All H-S and E-S components complete and valid.	2
Unistructural [U]	Able to complete some components only.	1
Prestructural [P]	There are pieces which makes no sense or the answer is totally wrong.	0
No attempt [N]	The answer is blank.	999

## Section C: Tracing

Q1:

```
4  int x;  
5  if(x<2)  
6      printf("%d",x);  
7  else if(x<4)  
8      printf("%d", 2*x);  
9      else if(x<6)  
10         printf("%d",3*x);  
11         else  
12             printf("%d", 4*x);  
13
```

- a) Which branch of **if/else** structure will be executed if  $x = 5$ .

*Sol: Line 9 will be executed.*

- b) What is the output of the program when  $x=6$ .

*Sol: 24*

Q2:

```
4  int i, j, x = 0;  
5  for (i = 0; i < 3; ++i)  
6  {  
7      for (j = 0; j < i; ++j)  
8      {  
9          x += (i + j - 1);  
10     }  
11 }  
12 printf("x = %d", x);
```

- a) Write the value of **x** when **i = 1**

*Sol:  $x=0$*

- b) Write the value of **x** at **line 12**.

*Sol:  $x=3$*

Table B.3 shows the SOLO classification for Tracing questions.

Table B.3

*SOLO Classification for Tracing Questions*

SOLO level	Indicator	Raw Score
Relational [R]	Able to provide correct solution to both parts a) and b).	3
Multistructural [M]	Able to provide solution to part a) and b) with minor errors or omissions.	2
Unistructural [U]	Only one part of the two is completed correctly.	1
Prestructural [P]	There are bits of pieces which makes no sense or the answer is totally wrong.	0
No attempt [N]	The answer is blank.	999

**Section D: Explaining**

Explain in plain English the purpose of the following code:

Q1:

```
4   int a =3;
5   int b = 7;
6   int c;
7
8   c = a;
9   a = b;
10  b = c;
11
12  printf("a = %d, b= %d ",a,b);
```

A suitable solution may look like this:

*The code swaps the values in a and b.*



Q2:

```
int i,c=0;
int a[10]= {2,3,4,5,7,8,9,12,34,6};

for(i=0;i<10;i++)
{
    if(a[i]%2==0)
    {
        c++;
    }
}
printf("%d",c);
```

A suitable solution may look like this:

*The code counts the number of even numbers in a given array.*

Table B.4 shows the SOLO classification for Explaining questions.

Table B.4

*SOLO Classification for Explaining questions*

SOLO level	Indicator	Raw Score
Relational [R]	Able to provide a summary of what the code does in terms of its purpose.	5
Relational Error [RE]	Able to provide a summary of what the code does in terms of its purpose but has some minor errors or omissions.	4
Multistructural [M]	Able to provide a line-by-line description of all the code.	3
Multistructural Error [ME]	Able to provide a line-by-line description of most of the code but has some minor errors or omissions.	2
Unistructural [U]	Able to provide description of one portion of the code.	1
Prestructural [P]	There are bits of pieces which makes no sense or the answer is totally wrong.	0
No attempt [N]	The answer is blank.	999

## Section E: Writing

Q1: Write a program in C that reads two integer number in the variables **x** and **y**. Divide **x** by **y** and report the result. You should confirm that **y** (divisor) is not zero before performing the division, as division by **0** is not possible and will cause your program to crash.

A suitable solution may look like this:

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int x, y;
6      float result;
7      printf("x and y(divisor):");
8      scanf("%d %d", &x,&y);
9
10     if(y==0)
11     {
12         printf("The divisor is zero. Try again!");
13     }
14     else
15     {
16         result = (float)x/y;
17         printf("The result of %d/%d = %f", x,y,result);
18     }
19
20     return 0;
21 }
```

The following components are examined to students' response to Writing Q1:

- Ability to declare variables of correct data types - Line 5 and 6 (E-U).
- Ability to read the variables from the console - Line 8 (E-S).
- Ability to check the divisor for zero and act accordingly - Line 10, 12 and 14 (E-S).
- Ability to compute correct division – Line 16 (H-S).
- Ability to show the desired output - Line 17 (E-S).
- Ability to write well-structured program in clear logical order (H-U).

Q2: Write a program in C that calculates the sum of every third integer, beginning with  $i = 2$  (i.e. calculate the sum of  $2 + 5 + 8 + 11 + \dots$ ) for all values of  $i$  that are less than 30.

A suitable solution may look like this:

```

5 |   int i,sum=0;
6 |
7 |   for(i=2;i<30;i=i+3)
8 |   {
9 |       sum+=i;
10 |   }
11 |   printf("The sum = %d", sum);

```

The following components are examined to students' response to Writing Q2:

- Ability to declare variables of correct data types - Line 5 (E-S).
- Ability to initialise a variable used to accumulate sum in the program - Line 5 where  $sum = 0$  (H-U).
- Ability to formulate correct loop - Line 7 (H-U).
- Ability to compute correct sum- Line 9 (H-S).
- Ability to show the desired output- Line 11(E-U).
- Ability to write well-structured program in clear logical order (H-U).

Table B. 5 shows the SOLO classification for Writing questions

Table B. 5

*SOLO Classification for Writing Questions*

SOLO level	Indicator	Raw Score
Relational [R]	Able to complete all of the components as a coherent whole.	6
Relational Error [RE]	Able to complete all of the components but has some minor syntax or logic errors or omissions.	5
Multistructural [M]	Able to complete most of the components. All H-S and E-S components complete and valid.	4
Multistructural Error [ME]	Able to complete most of the components but has some minor syntax or logic errors or omissions.	3

---

<b>SOLO level</b>	<b>Indicator</b>	<b>Raw Score</b>
Unistructural [U]	Able to complete some components only.	2
Unistructural error [UE]	Able to complete some components only with some syntax or logic errors.	
Prestructural [P]	There are bits of pieces which makes no sense or the answer is totally wrong.	0
No attempt [N]	The answer is blank.	999

---



## Appendix C: Sample Students' Responses to Programming Skills Test

### Questions Marked Using SOLO Classification

Figure C.1, Figure C.2 and Figure C.3 shows the sample students' response to algorithm design Q1

1. START  
 2. Declare variable a, b, c and x  
 3. Get the input from user and store it in a, b, c.  
 4. Compute their sum,  $d = a + b + c$   
 5. Compute  $x = \frac{d}{3}$   
 6. Print x as average of three numbers  
 7. STOP (End)

R-5

Start  
 Enter three numbers x, y and z.  
 Compute  $sum = x + y + z$ .  
 Compute  $average = sum / 3$ .  
 Print value of average.  
 Stop

a) omitted (E-U)  
 b) ✓ (E-S)  
 c) ✓ (E-S)  
 d) ✓ (E-S)  
 e) ✓ (E-U)  
 f) ✓ (E-U)  
 g) ✓ (H-U)

RE-4

Figure C.1. R and RE response to algorithm design Q1.

Enter the

i. Start  
 ii. Read the numbers  
 iii. Initialise  $Sum = a + b + c$   
 iv.  $average = \frac{Sum(a+b+c)}{3}$   
 v. display average.  
 vi. End

a) x (E-U)  
 b) ✓ (E-S)  
 c) ✓ (E-S)  
 d) ✓ (E-S)  
 e) x (E-U)  
 f) ✓ (E-U)  
 g) ✓ (H-U)

M-3

Section A: Flowchart / Program  
 ① Average of three number:  
 ① Start  
 ② Declare the three number x, y, z.  
 ③ Read the numbers  
 ④ Calculate the sum  
 $sum \leftarrow x + y + z$   
 ⑤ calculate the average  
 $Average = \frac{sum}{total\ number(n)}$   
 ⑥ End

a = ✓ (E-U)  
 b = ✓ (E-U)  
 c = ✓ (E-S)  
 d = x (E-S)  
 e = ✓ (E-U)  
 f = ✓ (E-U)  
 g = ✓ (H-U)

ME-3

Figure C.2. M and ME response to algorithm design Q1.

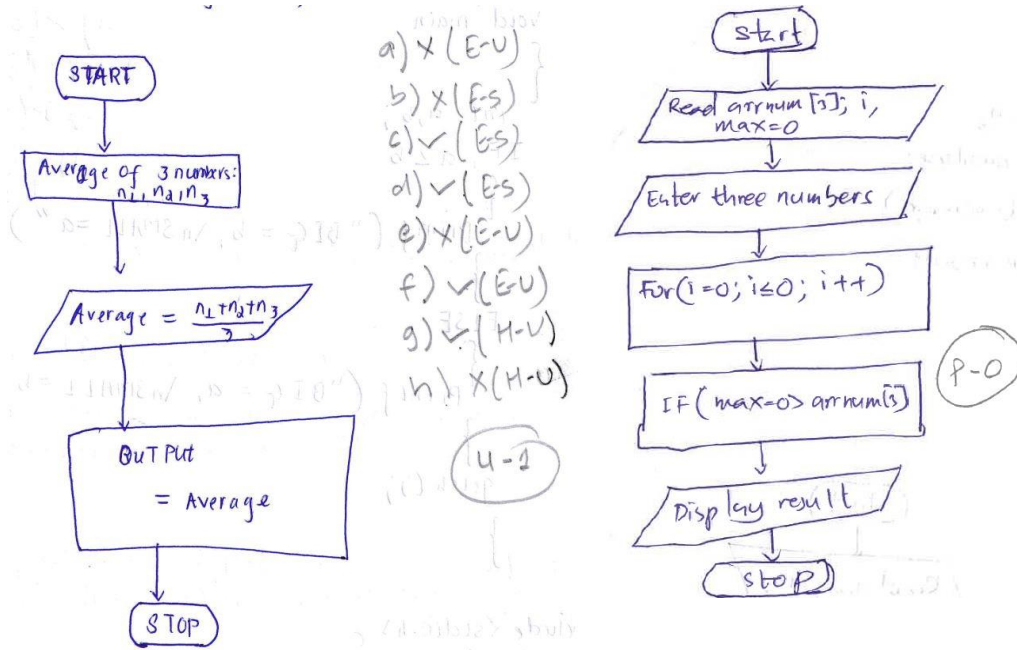


Figure C.3. U and P response to algorithm design Q1.

Figure C.4, Figure C.5 and Figure C.6 shows the sample students' response to algorithm design Q2

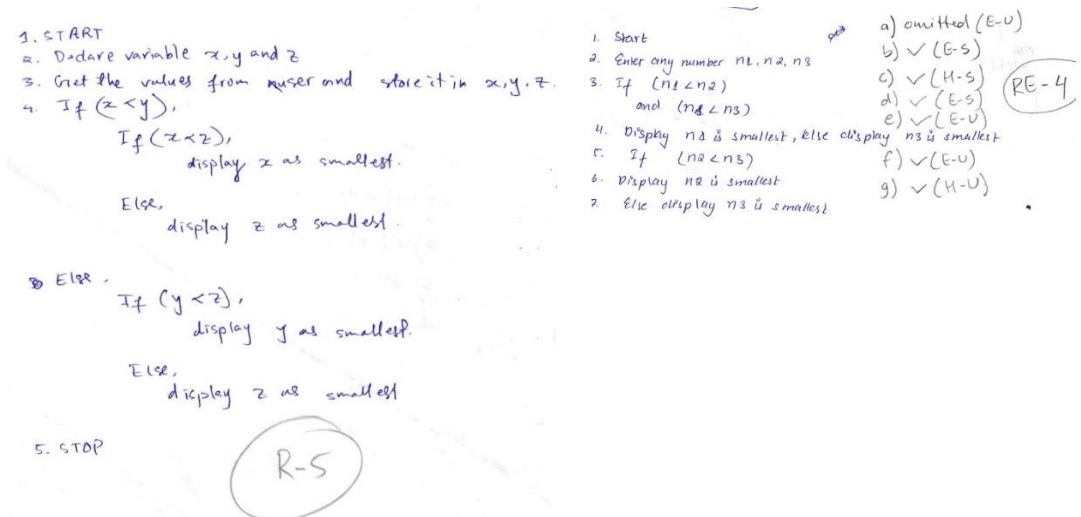


Figure C.4. R and RE response to algorithm design Q2.

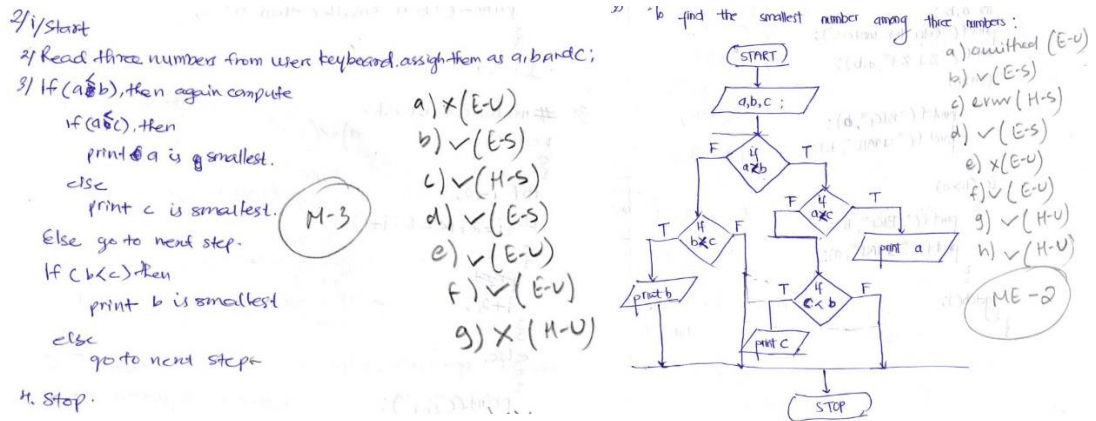


Figure C.5. M and ME response to algorithm design Q2.

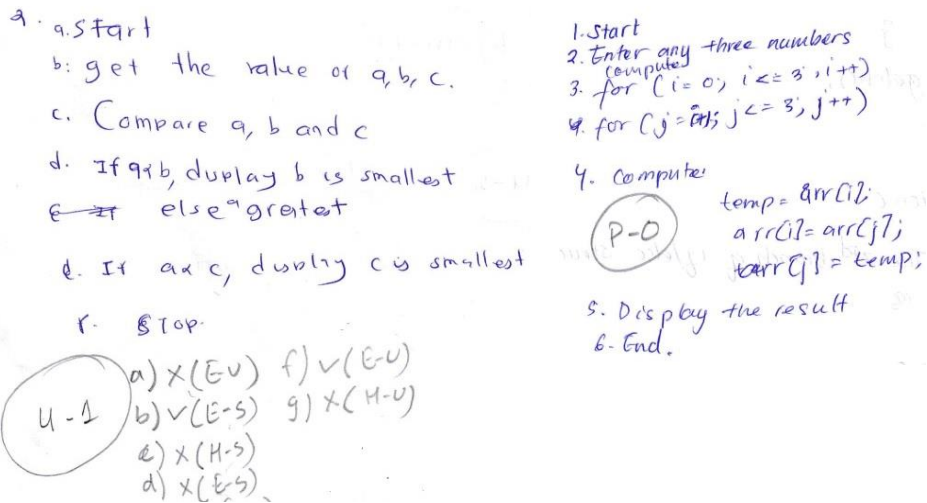


Figure C.6. U and P response to algorithm design Q2.



Figure C.7 and Figure C.8 shows the sample students' response to translating Q1.

```
#include <stdio.h>
main()
{
    int a, b;
    printf("Enter two numbers:\n");
    scanf("%d %d", &a, &b);
    if (a < b)
    {
        printf("BIG = %d\n", b);
        printf("SMALL = %d\n", a);
    }
    else
    {
        printf("BIG = %d\n", a);
        printf("SMALL = %d\n", b);
    }
    getch();
}
```

a) ✓ (E-S)  
 b) ✓ (E-S)  
 c) ✓ (E-S)  
 d) ✓ (E-S) **R-3**  
 e) ✓ (E-U)

```
#include <stdio.h>
main()
{
    int a, b;
    printf("enter the value of a, b:");
    scanf("%d %d", &a, &b);
    if (a < b)
    {
        printf("BIG = %d", b);
        printf("SMALL = %d", a);
    }
    else
    {
        printf("BIG = %d", a);
        printf("SMALL = %d", b);
    }
    printf("BIG, SMALL");
}
```

a) ✓ (E-S)  
 b) ✓ (E-S)  
 c) ✓ (E-S)  
 d) ✓ (E-S)  
 e) ✓ (E-U)

**M-0**

Figure C.7. R and M response to translating Q1.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int a, b;
    if (a < b)
    {
        printf("b is Big");
    }
    else
    {
        printf("a is Big");
    }
    getch();
}
```

a) ✓ (E-S)  
 b) omitted (E-S)  
 c) ✓ (E-S)  
 d) incomplete (E-S)  
 e) ✓ (E-U)

**U-1**

```
INITIALISE a, b
IF a < b THEN
    DISPLAY big = b
    small = a
ELSE
    big = a
    small = b
DISPLAY big & small.
```

**P-0**

Figure C.8. U and P response to Translating Q1.

Figure C.9 and Figure C.10 shows the sample students' response to translating Q2.

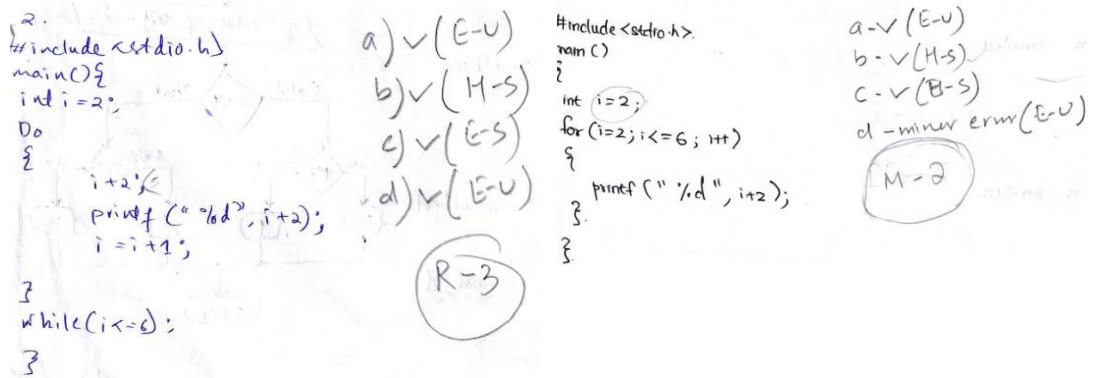


Figure C.9. R and M response to translating Q2.

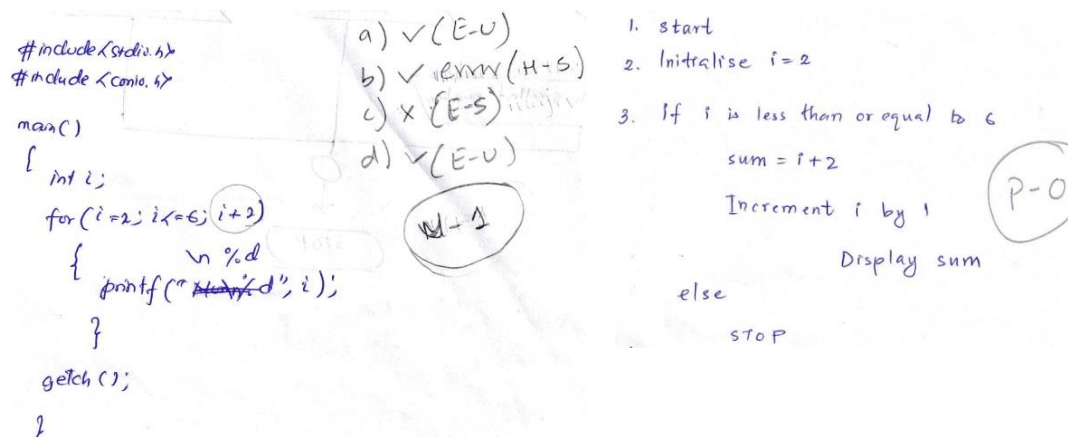


Figure C.10. U and P response to translating Q2.

Figure C.11 shows the sample students' response to tracing Q1.

a) 9<sup>th</sup> step i.e. else if ( $x < 6$ ) will be executed if  $x = 5$ .  
b) when  $x$  is 6, the output will be 24. (R-3)

a. else if part will be executed if  $x = 5$ . (M-2)  
b.  $4 * 6$  (the output when  $x = 6$ ).

a) else if branch will be executed  
b) Output will be (U-1)  
 $4 * x;$

a. All the branches of if/else structure will be executed if  $x = 5$ .  
b. When  $x = 6$ , the out-put is (P-0)  
612

Figure C.11. R, M, U and P response to tracing Q1.

Figure C.12 shows the sample students' response to tracing Q2.

a) Value of  $x$  when  $i=1$  is 0. ✓ (R-3)  
b) The value of  $x$  at line 12 is 3.

a) when  $i=1$ ,  $x=0$  ✓ (M-2)  
b) Value of  $x$  at line 12 is,  $x=1$

a)  $x=0$  (U-2)  
b)  $x=2$  ✗

a) The value of  $x$  when  $i=1$  is 2. (P-0)  
b) The value of  $x$  at line 12 is 2.

Figure C.12. R, M, U and P response to tracing Q2.

Figure C.13, Figure C.14 and Figure C.15 shows the sample students' response to explaining Q1.

The purpose of this code is to swap the  
value of  $a$  and  $b$ . (R-5)

The purpose of this code is to swap two  
number using third variable (RE-4)

Figure C.13. R and RE response to explaining Q1.

Variable ~~a~~ and ~~b~~ and ~~c~~ is declared as 3 and 7 respectively, while variable c is also declared. Value of c will then become ~~a~~ value of a, while b's value will be transferred to c. Value of c which is value of a will then be given to value of b. Then ~~a~~ and variable a and b are printed, resulting in exchange of values of a and ~~b~~ from 3 and 7 to 7 and 3 respectively. (M-3)

Here when it says  $c = a;$   
 $a = b;$   
 $b = c;$

it means that, c will take the value of a like  $c = 3;$   
 a will have the value of b i.e.  $a = 7;$   
 b will have the value of c i.e.  $b = 3;$

So, the result will be  $a = 7$  and  $b = 3.$   
 Since pointer is used here and the two values will be swapped.

(ME-2)

Figure C.14. M and ME response to explaining Q1.

The purpose of these code is to interchange the variables of code. (U-1)

int is a datatype which specifies whether the given value integer, floating or character type.  
 a, b and c are the containers / variables to store values.  
 printf is a function to print the output. (P-0)

Figure C.15. U and P response to explaining Q1.

Figure C.16, Figure C. 17 and Figure C.18 shows the sample students' response to explaining Q2.

The program is design for finding <sup>total</sup> number of (even numbers) in a given array. (R-5)

The purpose of this code is to print the even elements present in the array and to count how many even elements there are. (RE-4)

Figure C.16. R and RE response to explaining Q2.

2. The purpose of the following code is to display <sup>the number of</sup> even numbers stored in the variable a of array type which holds integer type ~~values~~ values currently. For loop is used to continuously check ~~for~~ each number in the array and if is used to check whether the element is even or odd. When the number is even the value of c gets incremented by 1 which is initially declared as 0. ~~At At~~ At the end the total number of even numbers is calculated and displayed. (M-3)

1). ~~initialise i and~~ Declare a variable i and initialise c=0. The datatype int contains an argument of maximum size of 10 numbers. ~~The~~ The program code is the code to find even numbers present in 10 sized integers from 2 to 34. If  $a[i] \% 2 == 0$ , which means if the remainder is zero then the number is even and it will get printed. (ME-2)

Figure C. 17. M and ME response to explaining Q2.

The following code is used to store the 10 different values of same data type and print all the even number. U-1

The purpose of this code is to interchange between variables and data it contains. P-0

Figure C.18. U and P response to explaining Q2.

Figure C.19, Figure C.20, Figure C.21 and Figure C.22 shows the sample students' response to writing Q1.

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    float x, y, division;
    printf("enter the values of x and y");
    scanf("%f%f", &x, &y);
    division = x/y;
    printf("division = %f", division);
    getch ();
    return 0;
}
```

a) ✓ (E-U)  
b) ✓ (E-S)  
c) ✗ (E-S)  
d) ✓ (H-U)  
e) ✓ (E-S)  
f) ✓ (H-U)

R-6

```
include <stdio.h>
main ()
{
    int x, y;
    float result;
    printf("Enter the value of x and y:");
    scanf("%d %d", &x, &y);
    if (y != 0)
    {
        printf result = x/y;
        printf("%f", result);
    }
}
```

a) ✓ (E-U)  
b) ✓ (E-S)  
c) ✓ (E-S)  
d) erwr (H-U)  
e) ✓ (E-S)  
f) ✓ (H-U)

RE-5

Figure C.19. R and RE response to writing Q1.

```

#include <stdio.h>
main() {
    int x, y;
    float z;
    printf("Enter two numbers");
    scanf("%d %d", &x, &y);
    if (y == 0)
    {
        printf("\n Infinity");
    }
    else
    {
        z = x/y;
        printf("x/y = %.2f", z);
    }
    getch();
}

```

a) ✓ (E-U)  
 b) ✓ (E-S)  
 c) ✓ (E-S)  
 d) ✓ (H-U)  
 e) ✓ (E-S)  
 f) ✓ (H-U)

M-4

```

#include <stdio.h>
main() {
    int x, y, z;
    printf("Enter any non-zero numbers");
    scanf("%d %d", &x, &y);
    z = x/y;
    printf("%d", z);
    getch();
}

```

a) ✓ minus error (E-U)  
 b) ✓ (E-S)  
 c) ✗ (E-S)  
 d) ✓ (H-U)  
 e) ✓ (E-S)  
 f) ✓ (H-U)

ME-3

Figure C.20. M and ME response to writing Q1.

```

#include <stdio.h>
main() {
    int n, y;
    float div;
    printf("Enter the value of n:");
    printf("Enter the value of y:");
    if (y != 0)
    {
        div = n/y;
        printf("%f", div);
    }
    else
    {
        printf("Division by 0 is not possible and will cause program to crash");
    }
    getch();
}

```

a) ✓ (E-U)  
 b) ✗ (E-S)  
 c) ✓ (E-S)  
 d) ✓ (H-U)  
 e) ✓ (E-S)  
 f) ✓ (H-U)

U-2

```

#include <stdio.h>
main() {
    int x, y, z;
    printf("Please enter any two numbers");
    scanf("%d %d", &x, &y);
    if (x % y == 0)
    {
        z = x % y;
        printf("%d", z);
    }
    Default
    printf("Reenter the numbers");
    getch();
}

```

a) ✓ (E-U)  
 b) ✓ (E-S)  
 c) ✗ (E-S)  
 d) ✗ (H-U)  
 e) ✓ (E-S)  
 f) ✓ (H-U)

UE-1

Figure C.21. U and UE response to writing Q1.



```

#include <stdio.h>
main() a) ✓(E-U)
{
  int x, y;
  y != 0;
  if (x % y == 1) P-0
  {
    printf("odd");
  }
  else
  {
    printf("even");
  }
  getch();
}

```

Figure C.22. P response to writing Q1.

Figure C.23, Figure C.24, Figure C.25 and Figure C.26 shows the sample students' response to writing Q2.

<pre> #include &lt;stdio.h&gt; main() {   int i, sum=0;   for (i=2; i&lt;30; i=i+3)   {     sum = sum + i;   }   printf("sum = %d", sum);   getch(); } </pre>	<p>a) ✓(E-U)  b) ✓(H-U)  c) ✓(H-S)  d) ✓(H-S)  e) ✓(E-U)  f) ✓(H-U)</p> <p style="text-align: center; border: 1px solid black; border-radius: 50%; padding: 5px;">R-6</p>	<pre> #include &lt;stdio.h&gt; main() {   int i, sum=0;   for (i=2; i&lt;=30; i=i+3)   {     sum = sum + i;   }   printf("The sum is %d", sum);   getch(); } </pre>	<p>a) ✓(E-S)  b) ✓(H-U)  c) minus even (H-U)  d) ✓(H-S)  e) ✓(H-U)</p> <p style="text-align: center; border: 1px solid black; border-radius: 50%; padding: 5px;">RE-5</p>
---	---	---	---

Figure C.23. R and RE response to writing Q2.

```
#include <stdio.h>
main() {
    int i, sum=0;
    for (i=2; i<30; i=i+3) {
        sum = sum + i;
    }
    printf ("The sum is %d", sum);
}
```

a) ✓ (E-S)  
 b) ✓ (H-U)  
 c) ✓ (H-U)  
 d) ✓ (H-S)  
 e) ✓ (E-U)  
 f) ✓ (H-U)  
 MI-4

```
main()
{
    int i, sum=0
    for (i=2; i<30; i++) {
        sum = sum + i;
    }
    i = i + 3;
    printf ("%d", sum);
}
```

ME-3

Figure C.24. M and ME response to writing Q2.

```
#include <stdio.h>
main()
{
    int i;
    i = 2;
    for (i=2; i<=30; i=i+3)
    {
        printf ("%d", i);
    }
    getch();
}
```

a) ✓ (E-S)  
 b) ○ (H-U)  
 c) ✓ (H-U)  
 d) ○ (H-U)  
 e) × (E-U)  
 f) ✓ (H-U)  
 U-2

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int i;
    for (i=2; i<30; i++)
    {
        i = 3 * i - 4;
        printf ("%d", i);
    }
    getch();
}
```

a) ✓ (E-S)  
 b) × (H-U)  
 c) ✓ (H-U)  
 d) × (H-S)  
 e) × (E-U)  
 f) ✓ (H-U)  
 UE-1

Figure C.25. U and UE response to writing Q2.

```

#include <stdio.h>
#include <conio.h>
void main()
{
    int x, y, sum; i=2, x, y, sum;
    printf("Enter the numbers: \n");
    scanf("%d %d %d", &x, &y, &sum);
    for(i=2; i<=3; i++)
    {
        sum = x+y;
        printf("P-O");
    }
    if (i>30)
        printf("Error");
    printf("%d = %d+%d", x, y, sum);
    getch();
}

```

Figure C.26. P response to writing Q2.

## Appendix D: Participant Consent Form

PARTICIPANT CONSENT FORM



Curtin University

**HREC Approval Number:** HRE2016-0318  
**Project Title:** Investigating Introductory Computer Programming and Student learning Outcomes in Royal University of Bhutan  
**Principal Investigator:** Dr. Martin Cooper  
**Co-Investigator:** Prof. David Treagust  
**Student Researcher:** Mani Pelmo

I have read the information provided in the Participant Information letter and understand the purposes of this study. I agree to participate in this study and I am aware that I can withdraw at any time without reason and without prejudice.

I understand that the information I provide is treated as strictly confidential and will not be released by the researcher in any form that may identify me.

I agree that research data gathered for the study may be published provided my name is not used.

Name:.....

Signature:.....

Date:.....



## Appendix E: Participant Information Letter

### PARTICIPANT INFORMATION STATEMENT



**HREC Approval Number:** HRE2016-0318  
**Project Title:** Investigating Introductory Computer Programming and Student learning Outcomes in Royal University of Bhutan

Dear Lecturer,

You are invited to participate in a study of key indicators of success in learning Introductory Computer Programming (CS1). Studies have shown that learning to program is not easy for many students. As such there are typically high failure rates in CS1. The purpose of this study is to improve CS1 learning outcomes by studying how students' learn and acquire programming skills as well as factors that contribute to the success of CS1. Therefore, this study aims to investigate the indicators of success for students' studying CS1 in Royal University of Bhutan. You are invited to participate in this study because you have taught Introductory Computer Programming at RUB.

This study is important for RUB and for Bhutan as the results from your participation seek to benefit students and lecturers in CS1 and provide a basis for further recommendations in curriculum reform to improve teaching and learning in CS1.

The research is being conducted by Mani Pelmo under the supervision of Dr. Martin Cooper and Professor David Treagust in Australia. The research is funded by a grant from Schlumberger Foundation, Faculty for the Future Fellowship program to obtain a Doctor of Philosophy at Curtin University.

There will be no cost to you and you will be given some incentives for participating in this project. You are invited to complete a survey form. A survey will take 10-15 minutes. Your participation in this research is entirely voluntary. You may withdraw at any stage without it affecting your rights or my responsibilities. When you have signed the consent form I will assume that you have agreed to participate and allow me to use your data in this research.

The information collected in this research will be kept confidential. The results of this research may be presented at conferences or published in professional journals. You will not be identified in any results that are published or presented.

Thank you very much for your participation in this research. Your participation is greatly appreciated.

Kind regards,

Mani Pelmo  
PhD Candidate  
mani.pelmo@postgrad.curtin.edu.au

## PARTICIPANT INFORMATION STATEMENT



**HREC Approval Number:** HRE2016-0318  
**Project Title:** Investigating Introductory Computer Programming and Student learning Outcomes in Royal University of Bhutan

Dear Student,

You are invited to participate in a study of key indicators of success in learning Introductory Computer Programming (CS1). Studies have shown that learning to program is not easy for many students. As such there are high failure rates in CS1. The purpose of this study is to improve CS1 learning outcomes by studying how you learn and acquire programming skills as well as factors that contribute to the success of CS1. Therefore, this study aims to investigate the indicators of success for students' studying CS1 in Royal University of Bhutan.

You are invited to participate in this study because you have studied Introductory Computer Programming in RUB. This study is important as the results from your participation seeks to benefit students and lecturers in CS1 and provide a basis for further recommendations in curriculum reform to improve teaching and learning in CS1. The research is being conducted by Mani Pelmo and is funded by a grant from Schlumberger Foundation, Faculty for the Future Fellowship program to obtain a Doctor of Philosophy at Curtin University.

There will be no cost to you and you will be given some incentives for participating in this project. You are invited to complete a survey form. It will take 15-20 minutes of your time. Your participation in this research is entirely voluntary. You may withdraw at any stage without it affecting your rights or my responsibilities. When you have signed the consent form I will assume that you have agreed to participate and allow me to use your data in this research.

The information collected in this research will be kept confidential. The results of this research may be presented at conferences or published in professional journals. You will not be identified in any results that are published or presented.

Thank you very much for your participation in this research. Your participation is greatly appreciated.

Kind regards,

Mani Pelmo  
PhD Candidate  
mani.pelmo@postgrad.curtin.edu.au

# Appendix F: Ethics Approval Letter

Office of Research and Development

GPO Box U1987  
Perth Western Australia 6845

Telephone +61 8 9266 7863  
Facsimile +61 8 9266 3793  
Web [research.curtin.edu.au](http://research.curtin.edu.au)

22-Sep-2016

Name: Mani Pelmo  
Department/School: Science and Mathematics Education Centre (SMEC)  
Email: [mani.pelmo@student.curtin.edu.au](mailto:mani.pelmo@student.curtin.edu.au)

Dear Mani Pelmo

**RE: Ethics approval**  
**Approval number: HRE2016-0318**

Thank you for submitting your application to the Human Research Ethics Office for the project **Investigating Introductory Computer Programming and Student learning Outcomes in Royal University of Bhutan**.

Your application was reviewed through the Curtin University low risk ethics review process.

The review outcome is: **Approved**.

Your proposal meets the requirements described in National Health and Medical Research Council's (NHMRC) *National Statement on Ethical Conduct in Human Research (2007)*.

Approval is granted for a period of one year from **22-Sep-2016** to **21-Sep-2017**. Continuation of approval will be granted on an annual basis following submission of an annual report.

Personnel authorised to work on this project:

Name	Role
Pelmo, Mani	Student
Rickards, Anthony	Supervisor
Treagust, David	Supervisor

## Standard conditions of approval

1. Research must be conducted according to the approved proposal
2. Report in a timely manner anything that might warrant review of ethical approval of the project including:
  - proposed changes to the approved proposal or conduct of the study
  - unanticipated problems that might affect continued ethical acceptability of the project
  - major deviations from the approved proposal and/or regulatory guidelines



- serious adverse events
3. Amendments to the proposal must be approved by the Human Research Ethics Office before they are implemented (except where an amendment is undertaken to eliminate an immediate risk to participants)
  4. An annual progress report must be submitted to the Human Research Ethics Office on or before the anniversary of approval and a completion report submitted on completion of the project
  5. Personnel working on this project must be adequately qualified by education, training and experience for their role, or supervised
  6. Personnel must disclose any actual or potential conflicts of interest, including any financial or other interest or affiliation, that bears on this project
  7. Changes to personnel working on this project must be reported to the Human Research Ethics Office
  8. Data and primary materials must be retained and stored in accordance with the [Western Australian University Sector Disposal Authority \(WAUSDA\)](#) and the [Curtin University Research Data and Primary Materials policy](#)
  9. Where practicable, results of the research should be made available to the research participants in a timely and clear manner
  10. Unless prohibited by contractual obligations, results of the research should be disseminated in a manner that will allow public scrutiny; the Human Research Ethics Office must be informed of any constraints on publication
  11. Ethics approval is dependent upon ongoing compliance of the research with the [Australian Code for the Responsible Conduct of Research](#), the [National Statement on Ethical Conduct in Human Research](#), applicable legal requirements, and with Curtin University policies, procedures and governance requirements
  12. The Human Research Ethics Office may conduct audits on a portion of approved projects.

**Special Conditions of Approval**

None

**This letter constitutes ethical approval only.** This project may not proceed until you have met all of the Curtin University research governance requirements.

Should you have any queries regarding consideration of your project, please contact the Ethics Support Officer for your faculty or the Ethics Office at [hrec@curtin.edu.au](mailto:hrec@curtin.edu.au) or on 9266 2784.

Yours sincerely



Dr Catherine Gangell  
Manager, Research Integrity

## **Appendix G: Student Survey Form**



## Student Survey Form



This questionnaire has a number of questions about your attitude towards your studies and your usual way of studying. There is no *right* way of studying. It depends on what suits your own style and the course you are studying. Accordingly, it is important that you answer each question as honestly as you can. Give the answer that would apply to the **Introductory Computer Programming** module.

Please read each of the following items completely and circle the one which accurately describes your feelings.

Each letter stand for the following response. "This item is ..."

A—*never or only rarely* true of me

B—*sometimes* true of me

C—true of me about *half the time*

D—*frequently* true of me

E—*always or almost always* true of me

1. I find that at times studying gives me a feeling of deep personal satisfaction.	A B C D E
2. I feel that virtually any topic can be highly interesting once I get into it.	A B C D E
3. I find that studying academic topics can at times be as exciting as a good novel or movie.	A B C D E
4. I work hard at my studies because I find the material interesting.	A B C D E
5. I come to most classes with questions in mind that I want answering.	A B C D E
6. I find that I have to do enough work on a topic so that I can form my own conclusions before I am satisfied.	A B C D E
7. I find most new topics interesting and often spend extra time trying to obtain more information about them.	A B C D E
8. I test myself on important topics until I understand them completely.	A B C D E
9. I spend a lot of my free time finding out more about interesting topics which have been discussed in different classes.	A B C D E
10. I make a point of looking at most of the suggested readings that go with the lectures.	A B C D E
11. My aim is to pass the course while doing as little work as possible.	A B C D E
12. I do not find my course very interesting so I keep my work to the minimum.	A B C D E
13. I find I can get by in most assessments by memorizing key sections rather than trying to understand them.	A B C D E
14. I find it is not helpful to study topics in depth. It confuses and wastes time, when all you need is a passing acquaintance with topics.	A B C D E
15. I see no point in learning material which is not likely to be in the examination.	A B C D E
16. I only study seriously what's given out in class or in the course outlines.	A B C D E
17. I learn some things by rote, going over and over them until I know them by heart even if I do not understand them.	A B C D E
18. I generally restrict my study to what is specifically set as I think it is unnecessary to do anything extra.	A B C D E
19. I believe that lecturers shouldn't expect students to spend significant amounts of time studying material everyone knows won't be examined.	A B C D E
20. I find the best way to pass examinations is to try to remember answers to likely questions.	A B C D E

THANK YOU FOR YOUR TIME

## Student Survey Form



### Investigating Introductory Computer Programming and Student Learning Outcomes at the Royal University of Bhutan

Thank you for your willingness to participate in the study of factors that may improve your learning outcomes in **Introductory Computer Programming module** (will be referred to as CS1). This survey forms part of Mani Pelmo's Doctoral research at Curtin University of Technology, Australia. Your individual responses to this survey will be kept confidential and will be used only for this research purpose. Please be honest and accurate as far as possible to enable the researcher to rely on the data given to analyse the computing science discipline.

## Student Survey Form



StudentID: \_\_\_\_\_ Gender (M/F): \_\_\_\_\_ College: \_\_\_\_\_ DOB: \_\_\_\_/\_\_\_\_/\_\_\_\_  
Today's Date: \_\_\_\_/\_\_\_\_/\_\_\_\_

Your programme:

- |   |  |
|---|--|
| <input type="checkbox"/> Diploma in Computer Hardware & Networking            | <input type="checkbox"/> B.Sc. Computer Application  |
| <input type="checkbox"/> B.E. Electronics and Communication Engineering       | <input type="checkbox"/> B.E. Civil Engineering      |
| <input type="checkbox"/> Diploma in Electronics and Communication Engineering | <input type="checkbox"/> B.E. Electrical Engineering |
| <input type="checkbox"/> B.E. Information Technology                          | <input type="checkbox"/> B.Sc. Computer Science      |
| <input type="checkbox"/> Bachelor of Architecture                             | <input type="checkbox"/> B.Sc. Physical Science      |
| <input type="checkbox"/> Other _____  |  |

1. Did you take any programming course prior to CS1 (formal class in high school or private training course)?

- Yes (go to question 2)                       No (go to question 3)

2. Check the programming languages that you have taken prior to CS1.

- |  |                               |                                 |
|--|-------------------------------|---------------------------------|
| <input type="checkbox"/> C                           | <input type="checkbox"/> Java | <input type="checkbox"/> Python |
| <input type="checkbox"/> C++                         | <input type="checkbox"/> C#   | <input type="checkbox"/> VB.Net |
| Any other programming language? Please specify _____ |                               |                                 |

3. How many hours per week on average did you spent on the following activities prior to CS1?

- \_\_\_\_ hrs/week - I used Internet to find information.  
\_\_\_\_ hrs/week - I played computer games (either online or offline).  
\_\_\_\_ hrs/week - I used application software such as Office, spreadsheet, presentation and database.

4. Indicate your high school (Year 12) marks that you scored in the following subjects:

Mathematics \_\_\_\_\_, Physics \_\_\_\_\_, Chemistry \_\_\_\_\_

5. Indicate your total marks in CS1 for the semester \_\_\_\_\_ (out of 100).

6. Which programming environment was used by the lecturers to teach CS1?

- |   |  |
|---|--|
| <input type="checkbox"/> Turbo C++                        | <input type="checkbox"/> Microsoft Visual Studio |
| <input type="checkbox"/> Terminal/Command-line            | <input type="checkbox"/> Dev-C++                 |
| <input type="checkbox"/> Not listed? Please specify _____ |  |

7. Is the programming environment used by your lecturer to teach CS1 user friendly or not?

- Yes                       No Any reason please comment \_\_\_\_\_

8. Did you use any other programming environment/s that you find it easy to use? If so list here-

\_\_\_\_\_

## Student Survey Form



9. In your experience and opinion, which programming language should you learn in CS1 classes?

- |   |                               |                                 |
|---|-------------------------------|---------------------------------|
| <input type="checkbox"/> C  | <input type="checkbox"/> C#   | <input type="checkbox"/> Python |
| <input type="checkbox"/> C++  | <input type="checkbox"/> Java | <input type="checkbox"/> VB.Net |
| <input type="checkbox"/> Any other programming language? Please specify _____ |                               |                                 |

10. What made you choose that language?

\_\_\_\_\_  
\_\_\_\_\_

11. Which teaching/learning methods and practices did you experience in CS1 that helped you learn in CS1?

- |   |   |
|---|---|
| <input type="checkbox"/> Pair/group programming         | <input type="checkbox"/> Live coding by the lecturer in class |
| <input type="checkbox"/> Any other please specify _____ |   |

12. In your experience in learning to program, order the following programming skills that you need to learn from first (1) to last (2/3/4/5). Write the same number on both the skills if you think that some programming skills needs to be learnt in parallel.

- \_\_\_\_\_ explaining a piece of code in plain English  
\_\_\_\_\_ drawing a flowchart/writing an algorithm  
\_\_\_\_\_ writing code  
\_\_\_\_\_ translating a flowchart/algorithm into code  
\_\_\_\_\_ tracing code

13. In your experience in learning to program, order the following programming skills in terms of contribution to the success in CS1 from highest (5) to lowest (1). Write the same number on both the skills if you think that some programming skills contributes equally to the success in CS1.

- \_\_\_\_\_ explaining a piece of code in plain English  
\_\_\_\_\_ drawing a flowchart/writing an algorithm  
\_\_\_\_\_ writing code  
\_\_\_\_\_ translating a flowchart/algorithm into code  
\_\_\_\_\_ tracing code

14. Any suggestions on how to improve teaching and learning of CS1 at RUB?

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

P.T.O

## **Appendix H: Lecturer Survey Form**



## Tutor Survey Form



14. In your experience in teaching CS1, order the following programming skills that students need to learn from first (1) to last (2/3/4/5). Write the same number on both the skills if you think that some programming skills needs to be learnt in parallel.

- \_\_\_\_\_ explaining a piece of code in plain English
- \_\_\_\_\_ drawing a flowchart/writing an algorithm
- \_\_\_\_\_ writing code
- \_\_\_\_\_ translating a flowchart into code
- \_\_\_\_\_ tracing code

15. In your experience in teaching CS1, order the following programming skills in terms of contribution to the success in CS1 from highest (5) to lowest (1). Write the same number on both the skills if you think that some programming skills contributes equally to the success in CS1:

- \_\_\_\_\_ explaining a piece of code in plain English
- \_\_\_\_\_ drawing a flowchart/writing algorithm
- \_\_\_\_\_ writing code
- \_\_\_\_\_ translating a flowchart into code
- \_\_\_\_\_ tracing code

16. Do you have any suggestions to improve teaching and learning of CS1 at RUB?

---

---

---

---

---

---

---

---

---

---

THANK YOU FOR YOUR TIME

## Tutor Survey Form



## Investigating Introductory Computer Programming and Student Learning Outcomes at the Royal University of Bhutan

Thank you for your willingness to participate in the study of factors that may improve your learning outcomes in **Introductory Computer Programming module** (will be referred to as CS1). This survey forms part of Mani Pelmo's Doctoral research at Curtin University of Technology, Australia. Your individual responses to this survey will be kept confidential and will be used only for this research purpose. Please be honest and accurate as far as possible to enable the researcher to rely on the data given to analyse the computing science discipline.



## Tutor Survey Form



Gender (M/F): \_\_\_ College: \_\_\_\_\_ Today's Date: \_\_\_/\_\_\_/\_\_\_

- Which programme have you taught in CS1:  
 Diploma in Computer Hardware & Networking  
 B.E. Electronics and Communication Engineering  
 Diploma in Electronics and Communication Engineering  
 B.E. Information Technology  
 Bachelor of Architecture  
 Other \_\_\_\_\_  
 B.Sc. Computer Application  
 B.E. Civil Engineering  
 B.E. Electrical Engineering  
 B.Sc. Computer Science  
 B.Sc. Physical Science
- Do you think it is beneficial for the students to have some programming experience prior to taking CS1?  
 Yes  No
- Would experience in any programming language give students an advantage in CS1? Please select one or more.  
 C  Java  Python  
 C++  C#  VB.Net  
Any other programming language? Please specify \_\_\_\_\_
- Do you think prior experience in any of these activities would assist your students in CS1?  
 Information search  Computer games  
 Application software (word, excel, PowerPoint and database)  
 Any other activities? Please specify here \_\_\_\_\_
- Please indicate minimum scores you would suggest is required for students prior to taking CS1 classes (in Standard 12).  
Mathematics \_\_\_\_, Physics \_\_\_\_, Chemistry \_\_\_\_
- Which programming paradigm do you think is suitable for CS1?  
 Procedural  Object Oriented  
 Any other? Please specify here \_\_\_\_\_

TUT2017

## Tutor Survey Form



- Which programming environment did you use to teach in CS1?  
 Turbo C++  Microsoft Visual Studio  
 Terminal/Command-line  Dev-C++  
 Not listed? Please specify \_\_\_\_\_
- Do you feel that students learn programming using the environment that you have chosen above?  
 Yes  
 No Any reason please comment: \_\_\_\_\_
- In your experience in teaching CS1, which programming environment is ideal to use to teach CS1?  
\_\_\_\_\_
- How can this environment (listed in question 9) help improve the student performance in CS1?  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_
- In your experience in teaching CS1, which programming language is ideal to teach in CS1?  
 C  C#  Python  
 C++  Java  VB.Net  
 Any other programming language? Please specify \_\_\_\_\_
- How can this programming language (chosen in question 11) improve student learning outcomes in CS1?  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_
- Which teaching/learning methods and practices do you think best suit students to learn programming?  
 Pair/group programming  Live coding in the class  
 Any other please specify \_\_\_\_\_

P.T.O

TUT2017

## Appendix I: Testing the Assumptions for Normality

### Assumption #1: The relationship between Independent Variables (IVs) and Dependent Variables (DV)

The first assumption of Multiple Regression is that the relationship between the IVs and the DV can be characterised by a straight line. The scatterplots shown in Figure I.1 checks the relationship between each of the IVs and DV.

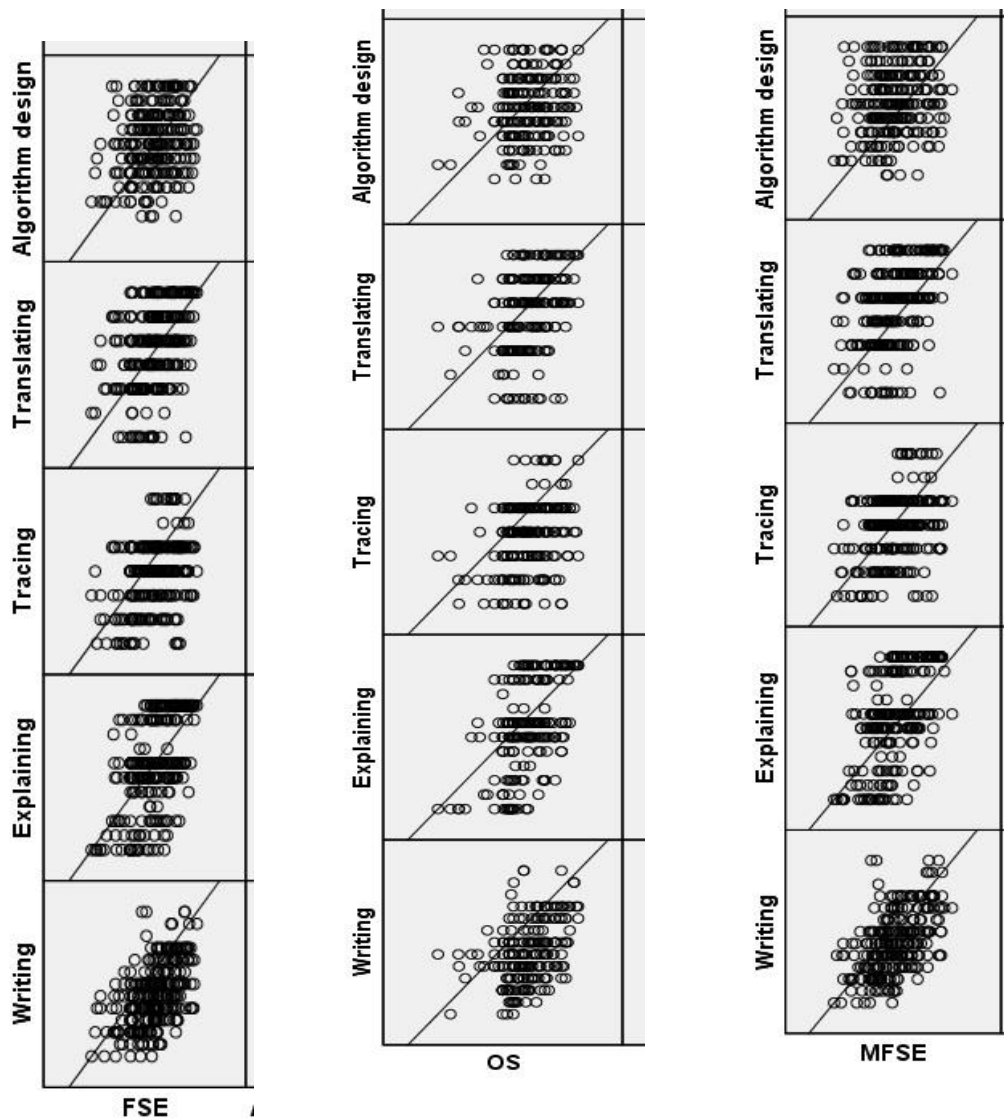


Figure I.1. Scatterplot showing the relationship between FSE and DV are linear.

**Assumption #2: There is no multicollinearity in the data.**

This is essentially the assumption that your predictors are not too highly correlated with one another. This assumption is verified by multicollinearity. First checking the values of correlation coefficients less than 0.8 as shown in Figure I. 2.

		Final Semester Exam	Semester Performance	Modified Final Semester Exam	Writing	Algorithm design	Translating	Tracing	Explaining
Final Semester Exam	Pearson Correlation	1	.855**	.955**	.522**	.252**	.415**	.353**	.555**
	Sig. (2-tailed)		.000	.000	.000	.000	.000	.000	.000
	N	285	215	276	272	283	284	282	280
Semester Performance	Pearson Correlation	.855**	1	.818**	.402**	.170*	.378**	.304**	.517**
	Sig. (2-tailed)	.000		.000	.000	.012	.000	.000	.000
	N	215	220	211	213	219	220	218	219
Modified Final Semester Exam	Pearson Correlation	.955**	.818**	1	.593**	.228**	.463**	.362**	.547**
	Sig. (2-tailed)	.000	.000		.000	.000	.000	.000	.000
	N	276	211	278	264	276	277	275	274

Figure I. 2. Correlation coefficient between Student Performance and programming skills.

In addition to that VIF scores are well below 10, and the tolerance scores to be above 0.2; which is the case with these variable as shown in the Figure I.3, Figure I.4, Figure I. 5 and Figure I. 6.

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.	Collinearity Statistics	
		B	Std. Error	Beta			Tolerance	VIF
1	(Constant)	27.629	2.501		11.046	.000		
	Algorithm design	.036	.031	.057	1.177	.240	.904	1.106
	Translating	.078	.027	.144	2.868	.004	.829	1.206
	Tracing	.083	.032	.129	2.614	.009	.849	1.178
	Explaining	.151	.025	.321	6.049	.000	.740	1.352
	Writing	.220	.035	.318	6.251	.000	.805	1.243

a. Dependent Variable: Final Semester Exam

Figure I.3. Collinearity Statistics highlighting tolerance and VIF (a)

**Coefficients<sup>a</sup>**

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.	Collinearity Statistics	
		B	Std. Error	Beta			Tolerance	VIF
1	(Constant)	41.730	2.474		16.866	.000		
	Algorithm design	.018	.030	.036	.622	.534	.934	1.070
	Translating	.066	.026	.157	2.600	.010	.840	1.190
	Tracing	.068	.029	.139	2.337	.020	.876	1.142
	Explaining	.126	.024	.342	5.294	.000	.737	1.357
	Writing	.110	.034	.201	3.263	.001	.815	1.226

a. Dependent Variable: Semester Performance

Figure I.4. Collinearity Statistics highlighting tolerance and VIF (b).

**Coefficients<sup>a</sup>**

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.	Collinearity Statistics	
		B	Std. Error	Beta			Tolerance	VIF
1	(Constant)	19.532	2.677		7.297	.000		
	Algorithm design	.018	.032	.027	.565	.573	.918	1.090
	Translating	.116	.028	.203	4.102	.000	.830	1.205
	Tracing	.104	.033	.154	3.170	.002	.861	1.162
	Explaining	.150	.026	.302	5.790	.000	.745	1.343
	Writing	.234	.037	.319	6.363	.000	.804	1.244

a. Dependent Variable: Modified Final Semester Exam

Figure I. 5. Collinearity Statistics highlighting tolerance and VIF (c).

**Coefficients<sup>a</sup>**

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.	Collinearity Statistics	
		B	Std. Error	Beta			Tolerance	VIF
1	(Constant)	13.944	4.274		3.262	.001		
	Algorithm design	.120	.052	.130	2.283	.023	.927	1.078
	Translating	.158	.046	.204	3.468	.001	.874	1.145
	Tracing	.042	.055	.046	.767	.444	.846	1.182
	Explaining	.163	.042	.244	3.921	.000	.782	1.278

a. Dependent Variable: Writing

Figure I. 6. Collinearity Statistics highlighting tolerance and VIF (d).

**Assumption #3: The values of the residuals are independent.**

This is basically the same as saying that we need our observations (or individual data points) to be independent from one another (or uncorrelated). We can test this

assumption using the **Durban-Watson** statistic. Figure I. 7 shows that the Durban-Watson statistic is between 0 and 4.

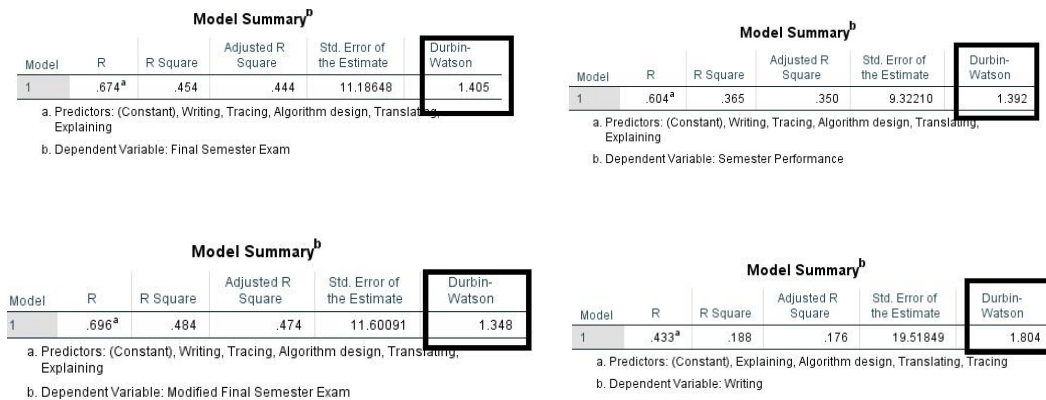


Figure I. 7. Durban-Watson statistic for the IVs.

**Assumption #4: The variance of the residuals is constant.**

This is called homoscedasticity, and is the assumption that the variation in the residuals (or the amount of error in the model) is similar at each point across the model. In other words, the spread of the residuals should be fairly constant at each point of the predictor variables (or across the linear model). To test the fourth assumption, we have plotted the standardised values of our model would predict against the standardised residuals obtained. From Figure I. 8, it appears to be more random than funnelled, this assumption is probably satisfied.

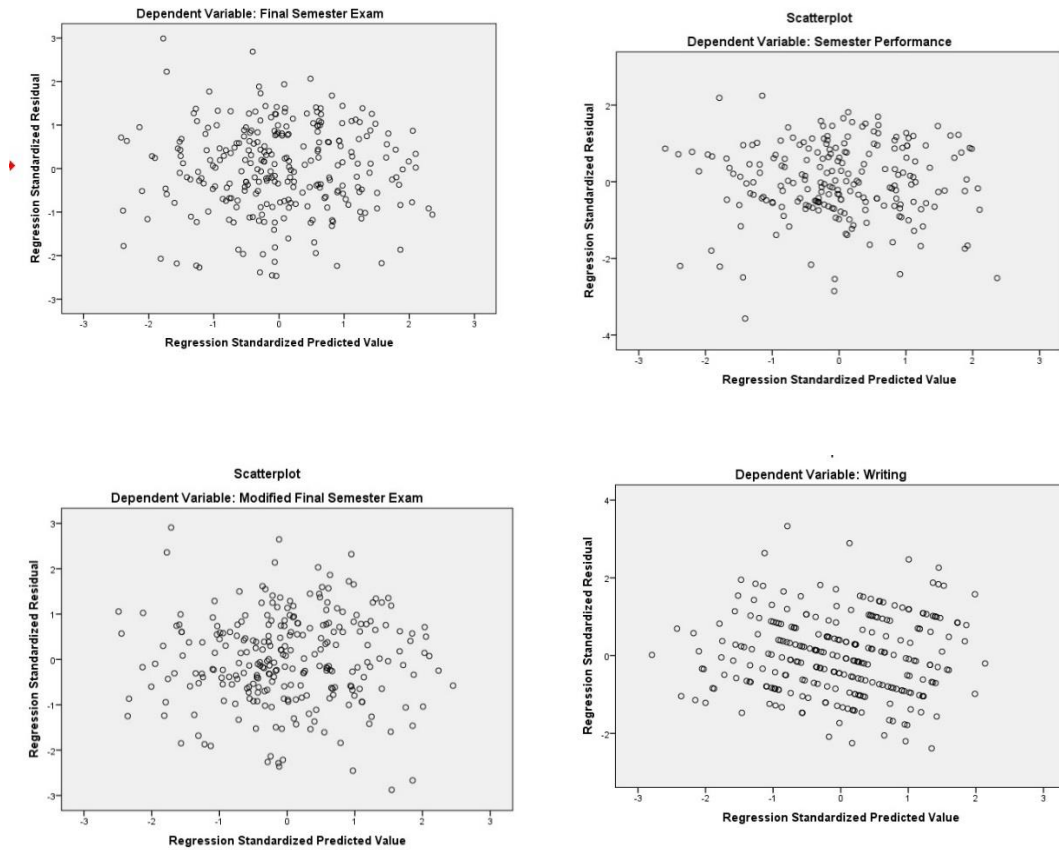


Figure I. 8. Scatter plot of standardised predicted value against residuals.

**Assumption #5: The values of the residuals are normally distributed.**

This can be tested by looking at the distribution of residuals. We can do this by checking the **Normal probability plot**. From Figure I.9, we can see that most of the dots lie closer to the diagonal line and residuals are distributed close to normal.

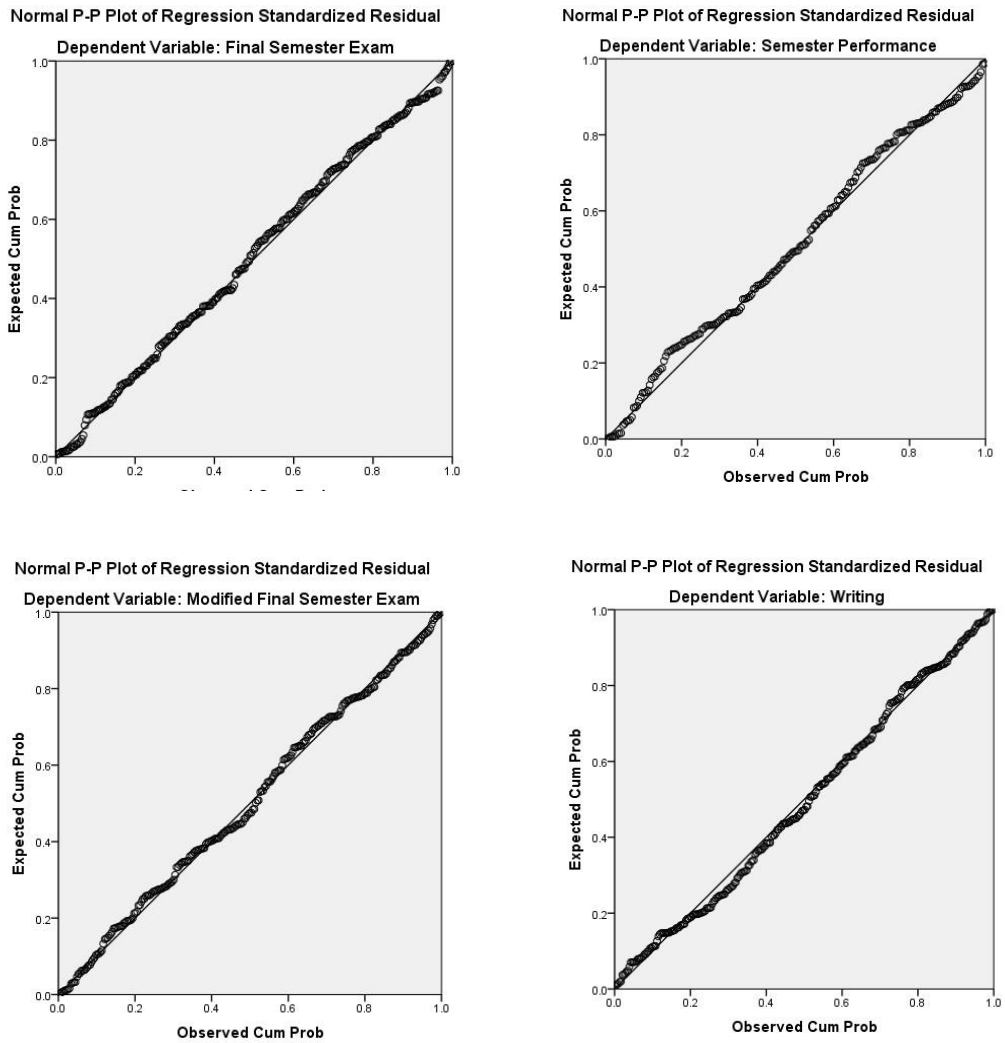


Figure I.9. P-P plots for the model with different DVs.

**Assumption #6: There are no influential cases biasing the model.**

The significant outliers and influential data points can place undue influence on our model which can be verified using **Cook's Distance**. From Figure I.10, we can see that each of the values are less than 1 showing there is not significant outliers.

	N	Minimum	Maximum	Mean	Std. Deviation
Cook's Distance FSE	268	.00000	.05135	.0039385	.00641527
Cook's Distance OS	212	.00000	.08392	.0050159	.00961570
Cook's Distance MFSE	261	.00000	.06580	.0042544	.00810582
Cook's Distance Writing	274	.00000	.02798	.0035862	.00526854
Valid N (listwise)	202				

*Figure I.10.* Cooks distance from the regression analysis with DVs namely FSE, OS, MFSE and Writing.





## Appendix J: Biggs Revised Study Process Questionnaire (R-SPQ-2F)

This questionnaire has a number of questions about your attitudes towards your studies and your usual way of studying.

There is no *right* way of studying. It depends on what suits your own style and the course you are studying. It is accordingly important that you answer each question as honestly as you can. If you think your answer to a question would depend on the subject being studied, give the answer that would apply to the subject(s) most important to you.

Please read each of the following items completely and circle the one which accurately describes your feelings. Each letter stand for the following response. "This item is ..."

A—*never or only rarely* true of me

B —*sometimes* true of me

C—true of me about *half the time*

D —*frequently* true of me

E— *always or almost always* true of me

Do not worry about projecting a good image. Your answers are CONFIDENTIAL. Thank you for your cooperation.

The responses to items are scored as follows:

A = 1, B = 2, C = 3, D = 4, E = 5

To obtain main scale scores add item scores as follows:

Deep Approach = 1 + 2 + 5 + 6 + 9 + 10 + 13 + 14 + 17 + 18

Surface Approach = 3 + 4 + 7 + 8 + 11 + 12 + 15 + 16 + 19 + 20

Subscale scores can be calculated as follows:

Deep Motive = 1 + 5 + 9 + 13 + 17

Deep Strategy = 2 + 6 + 10 + 14 + 18

Surface Motive = 3 + 7 + 11 + 15 + 19

Surface Strategy = 4 + 8 + 12 + 16 + 20

<b>Deep Motive</b>	<p>1. I find that at times studying gives me a feeling of deep personal satisfaction. A B C D E</p> <p>2. I feel that virtually any topic can be highly interesting once I get into it. A B C D E</p> <p>3. I find that studying academic topics can at times be as exciting as a good novel or movie. A B C D E</p> <p>4. I work hard at my studies because I find the material interesting. A B C D E</p> <p>5. I come to most classes with questions in mind that I want answering. A B C D E</p>
<b>Deep Strategy</b>	<p>6. I find that I have to do enough work on a topic so that I can form my own conclusions before I am satisfied. A B C D E</p> <p>7. I find most new topics interesting and often spend extra time trying to obtain more information about them. A B C D E</p> <p>8. I test myself on important topics until I understand them completely. A B C D E</p> <p>9. I spend a lot of my free time finding out more about interesting topics which have been discussed in different classes. A B C D E</p> <p>10. I make a point of looking at most of the suggested readings that go with the lectures. A B C D E</p>
<b>Surface Motive</b>	<p>11. My aim is to pass the course while doing as little work as possible. A B C D E</p> <p>12. I do not find my course very interesting so I keep my work to the minimum. A B C D E</p> <p>13. I find I can get by in most assessments by memorising key sections rather than trying to understand them. A B C D E</p> <p>14. I find it is not helpful to study topics in depth. It confuses and wastes time, when all you need is a passing acquaintance with topics. A B C D E</p> <p>15. I see no point in learning material which is not likely to be in the examination. A B C D E</p>
<b>Surface Strategy</b>	<p>16. I only study seriously what's given out in class or in the course outlines. A B C D E</p> <p>17. I learn some things by rote, going over and over them until I know them by heart even if I do not understand them. A B C D E</p> <p>18. I generally restrict my study to what is specifically set as I think it is unnecessary to do anything extra. A B C D E</p> <p>19. I believe that lecturers shouldn't expect students to spend significant amounts of time studying material everyone knows won't be examined. A B C D E</p> <p>20. I find the best way to pass examinations is to try to remember answers to likely questions. A B C D E</p>

## **Appendix K: Proposed CS1 Module Description**

**Module Title:** Introduction to Computer Programming

**Module code:** CS1

### **General Objectives**

This module introduces the fundamental principles of computer programming with an emphasis on problem solving strategies using structured programming techniques. The C programming language is used to introduce problem analysis, algorithm design and program implementation.

### **Learning Outcomes**

By the end of this module, students will be expected to

1. Analyse problems and derive their solutions.
2. Develop algorithms and draw the flow of logic for a given problem.
3. Write, compile, and debug simple computer programs for the given the problem statements.
4. Utilise a wide range of features available in C to write programs to solve problems.
5. Analyse problem requirements in order to understand what type of data and processes are involved in the system.
6. Design modular approach to satisfy those requirements.
7. Organize program code to implement the design.
8. Verify that the results obtained satisfy the original requirements.

### **Learning and Teaching Approach Used**

- |                   |                         |
|-------------------|-------------------------|
| <b>1. Lecture</b> | <b>4 hours per week</b> |
| a. Lecture        | 3 hours                 |
| b. Tutorial       | 1 hour                  |

**2. Practical** **3 hrs per week**

- a. Demonstration (Tutor) 1 hour
- b. Practice/Exercise (Student) 2 hours

**Assessment**

**1. Continuous Assessment** **50 Marks**

- a. Assignments 10 Marks
- b. Mid-Semester Examination 20 Marks
- c. Practical Examination and Viva 20 Marks

**2. Semester End Examination** **50 Marks**

**Subject Matter**

**Week 1:** Learning strategies

**Week 2, 3 & 4:** Algorithms and problem solving (tracing and explaining an algorithms)

**Week 5:** Introduce language features: data types, assignments, variable declaration, operators, expressions and simple input/output functions.

**Week 6:** Translate simple algorithms into programming code (translating). Using programming environment, write translated programs, compile, run and debug.

**Week 7, 8, 9 & 10:** Introduce language features: control structures, functions, arrays and files.

**Week 11 & 12:** Translate complex algorithms into programming code (translating). Manually execute the translated programs (tracing) and explain the purpose of the translated code (explaining).

**Week 13:** Introduce simple code writing from the given problem (writing) using programming environment. Manual tracing and explaining should go in parallel as well.

**Week 14:** Introduce complex code writing from the given problem (writing) using programming environment. Manual tracing and explaining should go in parallel as well.

**Week 15:** Introduce additional language features depending on what programming language is used in CS1 involving tracing, explaining and writing programs.



## Appendix L: CS1 FSE paper of three colleges at RUB

(This question paper contains *two* printed pages)

16PLT101A

Student No.....

**Semester End Examination, Autumn 2016**  
**B.Sc. Computer Science-Semester I**  
**Sherubtse College**  
**Royal University of Bhutan**  
**PLT101-Programming Fundamentals**

**Time: 3Hours**

**Max. Marks: 50**

*Write your student number on the top immediately on receipt of the question paper. All questions are compulsory; marks are given at the end of each question. Parts of a question should answered together. Spend the first 10 minutes in reading questions.*

1. Draw a flow chart to read temperature and display "it is hot today" if the temperature is greater than 32.0, otherwise display "It is normal today". Also write algorithm for same. [1+1]
2. Analyze the program written in 'C' and answer the questions given below program.

```
#include <stdio.h>
void main()
{
int array[3];
array[2]=5;
int i;
for(i=0; i <=2; i++)
{
printf("%d",array[i]);
}
return 0;
}
```

  - a) What will printf statement print if value of i=0.
  - b) What will be the output of program. [1+1]
3. What is a symbolic constant? What is a general form to declare a symbolic constant? How will you declare symbolic names TRUE and FALSE as 1 and 0 respectively. [1+1+1]
4. Write a program in 'C' using for ..loop to print the following structure  

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

[4]
5. What is a function? Differentiate between 'call by value' and 'call by reference' function?
6. Write a program to accept a number in main program, call a function to check a given number is prime or not using call by value.(For Example prime numbers are 2,3,5,7,.....divisible by 1 or itself)[1+1+3]



7. What is two dimensional Array? Write a program to add diagonal elements of matrix a[3][3] and print the sum at the end of program. **[4]**

8. What do you mean by a structure? How to define a structure in 'C'? Consider payment as a structure tag, (emp\_id, emp\_name, emp\_salary, month) are the members of a structure. Draw a figure how this structure will look like. Write a program in 'C' to accept the 5 value for record variable and display the accepted data. **[1+1+3]**

9. "Pointer represents the location rather than value", elaborate the statement. Give proper example how to declare pointer. Trace step by step and write output of a program

```
void main()
{
int u=55;
int v;
int *ptru;
int *ptrv;

ptru = &u;
v=*ptru;
ptrv=&v;

printf("\nu=%d      &u=%x      ptru=%x      *ptru=%d", u, &u, ptru, *ptru);
printf("\nv=%d      &v=%x      ptrv=%x      *ptrv=%d", v, &v, ptrv, *ptrv);
} [1+1+3]
```

10. What are different types of data files? Describe in detail stream oriented data file. Prepare a table

File Type	Meaning
-----	

Write a program in 'C' to create a data file(reservation.dat) for structure with 'bus' tag having members (passenger\_seat\_no,name,charges). Accept five passengers data for 'record' variable append data file. **[1+1+4]**

11. Write notes on  
a) C preprocessor  
b) Nested IF...ELSE **[2+2]**

12. Bhutan Power Corp.(BPC) charges for electricity consumptions per unit are  
First 100 units @ 1.28 Nu.  
101 to 200 units @ 2.45 Nu.  
If units 301 > @3.23 Nu.  
Write a program in 'c' to calculate total bill of a customer with parameters (consumer\_no, current\_reading, prev\_reading,energy\_consumed,total\_bill) **[5]**

13. Using while.. loop accept only odd numbers from 1 to 100 in an array. Print the same accepted odd numbers in reverse order from the array. **[5]**

---

Student No.

**ROYAL UNIVERSITY OF BHUTAN**  
**JIGME NAMGYEL ENGINEERING COLLEGE**  
**DEWATHANG:: BHUTAN**  
**AUTUMN SEMESTER EXAMINATION-2016**

**Class: D1CHN**

**Module: Fundamentals of Programming with C**

**Max. Marks: 40**

**Module Code : SYS101**

**Max. Time: 3 Hrs.**

*All the questions are compulsory*

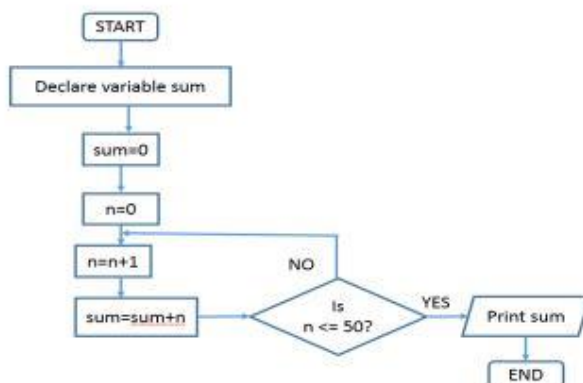
- 
1. Write down the appropriate option or answer in the answer script. **[4 x 0.25 = 1]**
- 1.1 When do we mention the prototype of a function?
- |              |                |
|--------------|----------------|
| a. Defining  | c. Prototyping |
| b. Declaring | d. Calling.    |
- 1.2 Which of the following is the correct usage of conditional operators in C?
- |                                    |   |
|------------------------------------|---|
| a. <code>a&gt;b? c=30:c=40;</code> | c. <code>max=a&gt;b? a&gt;c? a:c:b&gt;c?b:c;</code> |
| b. <code>a&gt;b? c=30;</code>      | d. <code>return (a&gt;b)?(a:b)</code>               |
- 1.3 C programs are converted into machine language with the help of\_\_\_\_\_.
- |               |                        |
|---------------|------------------------|
| a. an editor  | c. An operating system |
| b. a compiler | d. None of these.      |
- 1.4 Which of the following is not a valid identifier?
- |             |              |
|-------------|--------------|
| a. 2016Exam | c. Exam_2016 |
| b. _Exam    | d. Exam2016  |
2. Define the following terms. **[4 x 1=4]**
- 2.1 Interpreter  
2.2 Pseudocode  
2.3 Identifier  
2.4 Array
3. Write an algorithm or draw flowchart for the following questions. **[4 x 2 = 8]**
- 3.1 Read and store the integer value in array and find out the largest of values stored in array and print the element.
- 3.2 Read the temperature value from the user and display "It is hot today" if the temperature value is greater than 32.0 and display "It is cold today" otherwise.
- 3.3 Check whether a number entered by the user is prime or not. A prime number is a positive integer which is divisible only by 1 and itself. For example, 2,3,5,7,11,13..etc are prime numbers.
- 3.4 Reverse a given number and print both reverse and the given number.

4. Translate the following algorithm/flowchart into C language

[2+2 = 4]

- 4.1
1. Start
  2. Declare a variable n.
  3. Repeat the steps until n is not equal to -9.
    - 3.1. Read n
    - 3.2. If n is not equal to -9 then
      - 3.2.1. If n is divisible by 7 then  
Print "The number n is divisible by 7."
      - 3.2.2. Else  
Print "The number n is not divisible by 7."
    - 3.3. Else  
Stop repeating Step 3.
  - 4: End

4.2



5. Analyze the given segment of the code and answer the question that follows:

[4 x 1 = 4]

5.1.

```
int x;
if(x>3){
    if(x>4){
        printf("A");
    }
    else{
        printf("B");
    }
}
else if(x<2){
    if(x!=0){
        printf("C");
    }
}
printf("D");
```

- a) Which branch of if/else structure will be executed if x=1.
- b) What is the output of the program when x=4.

5.2.

```
int main()
{
    int array[3];
    array[2]=5;
    int i;
    for(i=0;i<=2;i++)
    {
        printf("%d",array[i]);
    }
    return 0;
}
```

- a) What is printed when i=0?
- b) What is the output of the program?

6. Explain in plain English the purpose of the following code:

[3 x 1 = 3]

6.1.

```
int x=2,n=3;
int i=1,r=1;
if(n==0)
{
    r=1;
}
else
    While(i<=n)
    {
        r=r*x;
        i++;
    }
printf("%d",r);
}
```

6.2.

```
int method(int arr[])
{
    int i;
    int num=0;
    for(i=0;i<5;i++)
    {
        num+=arr[i];
    }
    return num;
}
```

6.3.

```
#include<stdio.h>
int main()
{
    int a,b,c;
    int d;
    d=a>b?a>c?a:c:b>c?b:c;
    printf("%d",d);

    return 0;
}
```

7. Writing

[4+4+4+4= 16]

7.1 Briefly, explain the various steps to solve a problem by a computer.

7.2 With an example, explain jump statements in C language.

7.3 Explain four reasons to avoid goto statements in C programming.

7.4 Write a programme to print the following series.

```
1 2 3 5 6 8 10
10 9 8 7 6 5
5 10 15 20 25 30 35 40 45 50
27 24 21 18 15 12 9 6 3
```



Student No.

**ROYAL UNIVERSITY OF BHUTAN  
COLLEGE OF SCIENCE AND TECHNOLOGY  
PHUENTSHOLING: BHUTAN**

**WINTER SEMESTER EXAMINATION: 2016**

**Class : BE/BArch First Year**  
**Module : Introduction to Programming**  
**Module Code : CPL101**  
**Semester : I**  
**Max. Marks : 50**  
**Max. Time : 3 Hrs**

**General Instructions:**

1. *Answer all the questions.*

**Question No.1: Select the correct answer**

**[ 0.5X10=5]**

- 1.1 C language was developed at?  
A) Bell Laboratories of USA in 1972  
B) Bell Laboratories of USA in 1970  
C) Sun Microsystems in 1970  
D) Cambridge University in 1972
- 1.2 There are two types of loops namely entry and exit controlled loops. Which one is a selection statement?  
A) for  
B) while  
C) switch  
D) goto
- 1.3 The base 16 number 1125 converted to base 10 is?  
A) 253  
B) 269  
C) 538  
D) 1159
- 1.4 Select the type of file extension that the object file will have.  
A) .ob  
B) .obje  
C) .obj  
D) .object
- 1.5 What is the difference between the 5's in these two expressions?  
`int num[5];`  
`num[5]=11;`  
A) First is particular element, second is type  
B) First is array size, second is particular element  
C) First is particular element, second is array size  
D) Both specify array size
- 1.6 Write the correct answer for the following snippet:  
`int n=1;`  
`char name[10]= "thinley";`  
`do`  
`{`  
`puts(name);`  
`}while(n<0);`  
A) 1  
B) Name  
C) thinley  
D) 0

- 1.7 The result of 15 & 9 is:  
A) 24  
B) 15  
C) 10  
D) 9
- 1.8 Write the correct answer for the following snippet:  
`char s1[45]= "12345", s2[30]= "4567";`  
`printf("%s",strcpy(s1,s2));`  
A) 123454567  
B) 4567  
C) 12345  
D) None of the above
- 1.9 Which operator connects the structure name to its member name?  
A) - (dash)  
B) . (dot)  
C) <-  
D) Both (b) and (c)
- 1.10 A function that is not related to reading data from a file  
A) fgets  
B) fgetc  
C) fscanf  
D) fopen

**Question No.2: Find errors in the following programs. Rewrite the correct code.**

**[1X5=5]**

- 2.1 `void my array(int y)`  
`{`  
`printf("%d", *y);`  
`}`  
`void main();`  
`{`  
`float a[3]={2.0,3.3,8.5};`  
`my array(&a[1]);`  
`}`
- 2.2 `int i;`  
`i=1;`  
`do{`  
`print("%f",i);`  
`i+++;`  
`}while(i<=5)`

```

2.3 f(int a,int b)
    {
        int a=20;
        return a;
    }

```

```

2.4 switch(2)
    {
        case 1;printf("HOD:Tashi");break;
        case 2;printf("HOD:Yeshi");break;
        case 3;printf("HOD:Cheku");break;
        default:print("No department");
    }

```

```

2.5 main()
    {
        int i=0;
        for(i=+5, i<=12, i=i+3)
        {
            printf("%d",j);
        }
    }

```

**Question No.3: Write the output for the following questions [1X5=5]**

```

3.1 void main .()
    {
        int x;
        float y;
        y=x=7.5;
        printf("x=%d y=%f",x,y);
    }

```

```

3.2 int i ;
    for(i=1; i<10; i++){
        if(i==2)
        {
            continue;
        }
        else if(i>5)
        {
            break;
        }
        printf("\n%d",i);
    }

```

```

3.3 void main()
    {
        char arr[6]={'K','A','R','M','A'},*p;
        p=&arr[0];
        printf("\t%c",*(p+4));
        printf("\t%c",*(p+3));
        printf("\t%c",*(p+2));
        printf("\t%c",*(p+1));
        getch();
    }

```

```

3.4 switch('b')
    {
        case 'E':
        case 'e':printf("A and a\n");
        case 'b':
        case 'B':printf("B and b");
        case 'A':printf("\nA and a");
        case 'D':printf("D and d\n");
        default: printf("Other alphabets");
        break;
    }

```



```

3.5 main()
{
    int i,j,n=5;
    for(i=n;i>=1;i--)
    {
        for(j=1;j<=n;j++)
        {
            if(j<=n-i)
            {
                printf(" ");
            }
            else
            {
                printf("*");
            }
        }
        printf("\n");
    }
}

```

**Question No.4: Answer the following questions**

- 4.1 Convert  $(F02AD)_{16}$  into octal number system and  $(11001.101)_2$  into decimal number system. [2]
- 4.2 What is the single-precision representation of 347.625? [2]
- 4.3 Define the following: [2]
  - a. Typecasting
  - b. Structure
- 4.4 What is a function? Explain the difference between user defined function and library function with example each. [3]
- 4.5 What do you understand by recursion? Give one simple example to illustrate recursion. [3]
- 4.6 What is a pointer? Name the two commonly used operators with pointer. [2]
- 4.7 Why should you prototype or have function declaration? [1]

**Question No.5: Write Program for the following questions**

- 5.1 Write pseudocode and draw a flowchart to compute the sum of all even numbers between 0-20. And write the program to solve the given problem. [6]
- 5.2 Write a program to design a simple calculator to perform basic calculation using switch case statement. Provide Options 1 to Add, 2 to Multiply, 3 to subtract and 4 to divide. Get two numbers and perform the basic arithmetic calculation. [3]
- 5.3 Write a function each to swap two numbers using call by value and call by reference concept. [3]
- 5.4 Write a program to compute the sum of the series  $1/1+2/(1+2)+3/(1+2+3)+\dots+N/(1+2+3+\dots+N)$  [2]
- 5.5 Write a program to add two 3x3 matrix in C. The values in both the matrices are provided by user. [3]
- 5.6 Write a program using structure to read and display the information of three students as shown below: [3]

ROLL NO	NAME	AGE	FATHER'S NAME	MOTHER'S NAME	VILLAGE
1	Karma	22	Dorji	Sangay	Paro
2	Dema	24	Pasa	Kinley	Haa
3	Dolo	20	Dawa	Dema	Gasa