Competency based learning framework of Introductory Programming course to enhance learner's motivation and skills

Muhammad Usama Ijaz[1], Maida Shahid[2] & Talha Waheed[3*]

Journal of Educational Psychology and Pedagogical Sciences (JEPPS)

ISSN:2791-0393 (Print) eISSN: 2791-0407

Vol.5, No. 1, (Jan-June, 2025): 59-86

[1]MPhil Scholar, Department of Computer Science, University of Engineering and Technology, Lahore, Pakistan. Email: mirza.osama96@gmail.com

[2] MPhil Scholar, Department of Computer Science, University of Engineering and Technology, Lahore, Pakistan. Email: maida.shahid@uet.com

[3*]Corresponding Author. Assistant Professor, Department of Computer Science, University of Engineering and Technology, Lahore, Pakistan. Email: twaheed@uet.com. (https://orcid.org/0000-0003-3825-5810).

Submitted 11 Jan 2025

Accepted 12 June 2025

Published 30 June 2025

DOI: https://doi.org/10.52587/jepps.v5i1.110

https://jepps.su.edu.pk/article/49

ABSTRACT

Computers have revolutionized business, education, government, commerce, and research, creating rapidly expanding career opportunities in computer science and related fields. As a result of these technological breakthroughs, computer science-related jobs are growing at a fast pace. However, computer science graduates have a high unemployment rate despite a significant need for computing and technology experts. This high level of unemployment is due to the discrepancies between the concepts taught in computer science degree programs and the skills required in the software industry. To fill this skill gap, Computing Machinery (ACM) and IEEE Computer Society (IEEE-CS) have devised guidelines for a new computing curriculum of BS (Computer sciences/IT) 1st semester in 2020 that has shifted the trend from Knowledge-based Learning to Competency-based learning. Many researchers have discussed skills and competencies, but they are not designing the programming courses by specifically targeting the skills needed in the IT industry. Therefore, this research aims to enhance the skills of students to match the skills needed in the IT industry by following competency-based learning among BS students. We have designed the introductory programming course based on three mega competencies that the software industries value. These are further divided into 8 mini competencies and sixty-two mini competencies to be easily taught in a 2-3 hour lecture duration. We also present our teaching experience with undergraduate computer science students at the University of Engineering and Technology, Lahore, Pakistan. Results show that competency-based learning had a positive impact on increasing students' motivation and improving their programming skills.

Keywords: Competency-based learning, competencies, 4C/ID model, programming skills, Undergraduate computing course, teaching framework

1. Introduction

With the advancements in technology, computers have conquered almost every aspect of our daily lives (Aspray, 2013). From our homes to our work, from our studies to our entertainment

devices, the use of computers is everywhere (Hennig-Thurau et al., 2021; Voskoglou, 2021). This intense use of computers has opened up a huge portal of job and career opportunities. The Bureau of Labor Statistics estimated that the computing job openings will increase by 12% in the US by 2024 (Labor, 2017). With this huge number of job openings, students have started taking admission in IT and Computer Science degree programs. However, most students enrolled in the universities drop out of their degrees without completion. At the same time, the students who do complete their degrees have a very hard time securing a good job (Bennedsen & Caspersen, 2007; Petersen et al., 2016; Watson & Li, 2014). Therefore, the need for software engineers continues to rise.

Traditionally, universities follow the Knowledge-Based Learning (KBL) approach (Whitehall & Lu, 1994), which has served a lot in teaching different subjects. However, with the demanding skills and competencies, KBL failed to deliver according to the expectation (Clear et al., 2019). The primary reasons for that are: focus on content rather than on students, lack of motivation, and less or no focus on developing skills and competencies. In the KBL approach, the students are bombarded with a huge amount of knowledge in the form of lectures and slides, more knowledge than they need. Students just cram this knowledge to pass their exams. This cramming does help the students secure their degrees. However, these students with an immense reservoir of knowledge struggle to implement that knowledge. Thus, the students are left with only knowledge but no skills (Begel & Simon, 2008).

Around 60% of US employers reported job openings that stay vacant for up to three months, and 81% of IT employers stated that their fresh employees are deficient in many skills, in which critical thinking and analytical reasoning are on top (McGlochlin, 2018). In this technological era, employers demand a fresh employee with an impeccable set of skills, not just knowledge of those skills (Carnevale & Smith, 2013; Hetling et al., 2014). This gap between the employers' demanded skills and those the students acquire is known as the 'Skills Gap' (Kim et al., 2006).

Another problem with KBL is that the students face difficulty visualizing the bigger picture. Therefore, they lose focus and motivation without the implementation of that knowledge (Cheah, 2020). They are unable to correlate their classroom learning with the real-life problems that are solved in the IT industry. Thus, they do not know how to use this newly acquired knowledge to excel in their professional careers. This puts the students under a psychological strain that drains their confidence and motivation, leaving them with anxiety and self-doubt (Tan et al., 2009).

In order to resolve these issues, the Association for Computing Machinery (ACM) and IEEE Computer Society (IEEE-CS) has provided a Computing Curricula based on competency-based learning that contains the paradigms for Global Computing Education (Force, 2020). This curriculum provides guidelines to computing instructors worldwide on effectively designing the competencies for computing courses. The basic idea behind this is to teach the most commonly used skills in the IT industry to the students one by one, rather than just the bucket full of knowledge. The Harvard University Competency Dictionary (Harvard University Competency Dictionary FY14, 2014) defines competency as:

"Competency composes an expanded perspective on education that augments knowledge (knowing what), with its skilled application (knowing how), motivated by the purpose (knowing why) to accomplish a task."

Each of these competencies comprises three components; knowledge, skill, and vision of a task. The knowledge represents the KNOW-WHAT of the task, i.e., a student knows what techniques she would need to solve that task. The skill represents the KNOW-HOW, i.e., how will she use her acquired knowledge to solve that task? Furthermore, the vision is simply the

KNOW-WHY, i.e., why she is doing it? A normal student can only be called a competent student if she possesses all three of these components.

Fulfilling the need of the hour, the Skills Framework for the Information Age (SFIA) stipulates the globally acknowledged competencies and skills required in the IT industry. The recent version of SFIA framework (SFIA 8) mentions 102 skills and competencies that the IT industry values (von Konsky et al., 2016).

Usually, a skill provided by SFIA is very general and can be distributed over different subjects. So, we have to cover all those subjects before trying to achieve a single skill. While first studying those subjects, students lose sight of the bigger picture of how these subjects are interlinked; thus, their motivation diminishes. Also, most of the talked about the skills and competency researchers (Kaharuddin, 2020; Rojas-López, 2019; Sultana, 2016). However, there is no proper road map for the instructors on designing the courses and preparing their lectures by following competency-based learning. The researchers who have designed computing courses have done so without the key element of competency-based learning (Gavrilović et al., 2018).

Thus, this research aims to improve the skills of CS graduates such that they match the skills required by the IT employers and increase the motivation of the students by teaching the introductory programming course based on competency-based learning.

This paper focuses on answering the following research questions:

What could be possible competencies for the introductory programming fundamentals course.

How can we divide these competencies into the mini and micro competencies for delivering the contents ideally in a lecture, and how to assess those competencies.

To address these research questions, we divide these competencies into three categories.

Mega Competencies (Level 3)

Mini Competencies (Level 2)

Micro-Competency (Level 1)

Introductory programming course consists of three mega competencies that the IT industry values. These mega competencies are further divided into eight mini competencies, which are further divided into sixty-two micro competencies. It is much easier for the instructor and the student to cover a micro competency in a single lecture. These micro competencies consist of vision (KNOW-WHY) (i.e., why study the following concepts), knowledge (KNOW-WHAT) (i.e., what concepts to study), and skill (KNOW-HOW) (i.e., how to use the concepts to solve the real-world problems). So, every micro competency induces knowledge of one skill into the students, and they get hands-on practice to acquire that skill. After covering certain micro competencies, a mini competency is achieved. Then, after covering certain mini competencies, mega competency is achieved, which helps achieve the Computer Science degree. This approach is distinguished from the rest as it keeps the students' interest intact the whole time by showing them the broader picture. The students know beforehand what they will achieve by the end of every micro competency and develop the required skills along the way. This improves the students' learning skills, confidence, motivation, and communication with their employers in the long run.

The rest of the paper is as follows. Section 2 provides the systematic literature review. Section 3 describes the proposed framework. In section 4, we describe the experimentation. Section 5 provides the experimentation results, and section 6 discusses the results.

2. Literature Review

Advancements in the field of pedagogy have always been the concern of the teaching community (Sobral, 2021). Researchers around the globe have strived for new and better approaches in this field. For decades, Association for Computing Machinery (ACM) has been working to improve the field of Computing (Dziallas & Fincher, 2015) and providing new teaching curriculum guidelines according to the institutional goals and needs. The Knowledge-Based Learning (KBL) Computing Curricula 2005 (Shackelford et al., 2006) proposed curriculum guidelines for five computing programs: Computer Engineering, Software Engineering, Computer Science, Information Systems, and Information Technology. It introduced a structure of Knowledge Units (KU) and Learning Outcomes (LO). This approach targets students' existing knowledge and builds up new knowledge on top of it (Geissler et al., 2020). The majority of the educational institutes in the world still use Knowledge-Based Learning to produce graduates (Longenecker et al., 2015).

ACM continued to develop computing curricula for CS in 2008 (Cassel et al., 2008) and computing curricula for CS in 2013 (Draft, 2013). Despite teaching courses designed on the latest guidelines provided by ACM, instructors found deficiencies in students' programming skills. Therefore, researchers continued to experiment with different teaching methodologies to overcome the students' lack of skills. Widely opted teaching methodologies for introductory programming courses include Active Learning (Barak et al., 2007), Pair Programming (Hannay et al., 2009), Flipped Classroom (Souza et al., 2015), and Project-Based Learning (Havenga, 2015). Each of these techniques induced interest and motivation of the student in programming, decreased the dropout ratio of the students but was unable to produce the industry-required skills in the students (Sobral, 2021).

Many authors proposed different teaching methodologies for teaching programming to novice programmers. The study by Woodley and Kamin (2007) argued the need for reforms in the teaching methodologies by providing a study for improving programming skills in undergraduates. The authors proposed a programming studio environment (like that of an art class) in which the instructor will give a one-hour lecture, followed by two hours discussion with the students. The study's experiment continued for two years and claimed that the students' skills significantly increased throughout the course.

The study by Liu and He (2015) proposed "business-oriented education." They sought help from the IT employers to give the recommended training to undergraduate students so that they might perform better in the industry. The study resulted in the professional computer education of the students. All these studies revealed that the KBL approach was insufficient to match the skills required by the IT industry.

The Skills Framework for Information Age (SFIA) provides a framework of skills defined by the collaboration of the IT industry employers (Brown, 2020). Some of the skills mentioned in SFIA 8 framework are programming/software development, software design, systems software, data modeling and design, data management, database design, database administrator, testing, information management, information security, quality assurance, data science, high-performance computing and machine learning cover the Computer Science degree program. The skill of Programming/Software Development is as follows:

"Designs, codes, verifies, tests, documents, amends and refactors programs/scripts."

This is a very general skill, and it could only be covered during a series of different computer science courses such as programming fundamentals, object-oriented programming, data structures, software development, and software engineering.

The curriculum for Information Communications and Technology (ICT) based on the framework given by SFIA (Konsky et al. 2016). The results of the study showed the positive impact of IT employers in defining the required skills in the curriculum. The results confirmed the need for skills and competencies in designing the IT-related programming curricula.

There are three major components of a competency: 'knowing-what' of knowledge, second is the 'knowing-how' of that knowledge (how to implement that knowledge), and the final is the 'know-why' (what can be accomplished after acquiring that knowledge). These three components are crucial in the student's learning and to make him/her stay focused during this learning. To Fulfil the growing demand for skills, ACM presented new guidelines based on competency-based learning (CBL) in the 2020 computing curricula (Clear et al., 2019). This curriculum provided guidelines to the computing instructors worldwide on effectively designing the competencies for computing courses. The basic idea is to teach the most commonly used skills in the IT industry to the students one by one, rather than just a bucket full of knowledge.

The initial learning scenarios of the Programming Fundamentals course was designed by Rojas-López et al. (2019). The main focus was to enhance the motivation and skills of the students. The idea was to develop the students' computational thinking by providing a course that mapped computational thinking skills to programming knowledge. The study targeted three trends, i.e., recognition of previous skills, training of new competencies, and preventive actions of the instructor's communication when a student lacks a specific skill. However, the study introduced the competencies at a higher abstraction level that might hinder the students' motivation towards learning.

While designing an informatics curriculum for K-12 education, constructivism (learning by doing) and critical thinking (the creative process of development) was focused (Dagiene et al. 2021). The study presents the curriculum contents for four subjects: Algorithm Design, Robotics, Programming, and Communication in Networks. Each subject consisted of around ten competencies, but these merely instructed the students on the "what" and "how" of the development phase. The study failed to convey the 'why' behind development to the students and, consequently, did not show them the broader picture of what they could achieve with these competencies. This led to the students' loss of motivation, which meant they needed to continue on the path of learning during the course.

Zhang. et al. (2020) conducted a study to validate how computational thinking skills are taught to K-9 students with the help of programming. The study's primary objective was to observe the progression of computational thinking (CT) skills throughout a student's compulsory education. The authors divided the CT skills according to the students' grades: 1-3, 4-6, 7-9. They found out that the skills devised by the authors were only grasped, in a better fashion, by the students of the third group. However, the authors were unable to implement the CT concepts into the CT skills, i.e., they provided the CT concepts to the students but did not validate those concepts in the study.

Although these studies incorporated competencies into their curriculum designs, they failed to provide a holistic view of the competencies that encompassed the entire undergraduate degree program in Computer Science.

At Western Governors University (WGU), USA, the university council defined a high-level set of competencies for each degree. After that, a group of contracted domain experts broke down the ten or so high-level competencies for a particular qualification into about thirty more specific competencies, around which the online courses to develop mastery of each competency were built.

Graduates' competencies were based on what they should know in the job and as professionals in their chosen field. Assessments were meant to evaluate mastery of each ability; as a result, students receive a pass/fail grade after each assessment. A degree is given when all thirty required skills are met (Marcus, 2017).

For the introductory programming course, they covered the following competencies:

1. Begin your course by discussing your course planning tool report with your instructor and creating your personalized course plan together.
2. The graduate examines basic computer programming elements, including data types, constants, variables, operators, and expressions.
3. The graduate determines how to achieve programming goals through functions and control structure.
4. The graduate interprets algorithms.
5. The graduate describes the steps of the software design process.
6. The graduate compares various scripting and programming languages.

These online degree programs have made great progress with nearly 40,000 students because they just taught the specific subjects to the students in each degree. These competencies are very general and do not cover the vision of what the students will achieve or produce after completing the specific course. Moreover, these so-called competencies are similar to learning objectives defined in knowledge-based learning. Furthermore, these degree programs ignore the importance of social learning, and these work well with some learning environments but lesser with others.

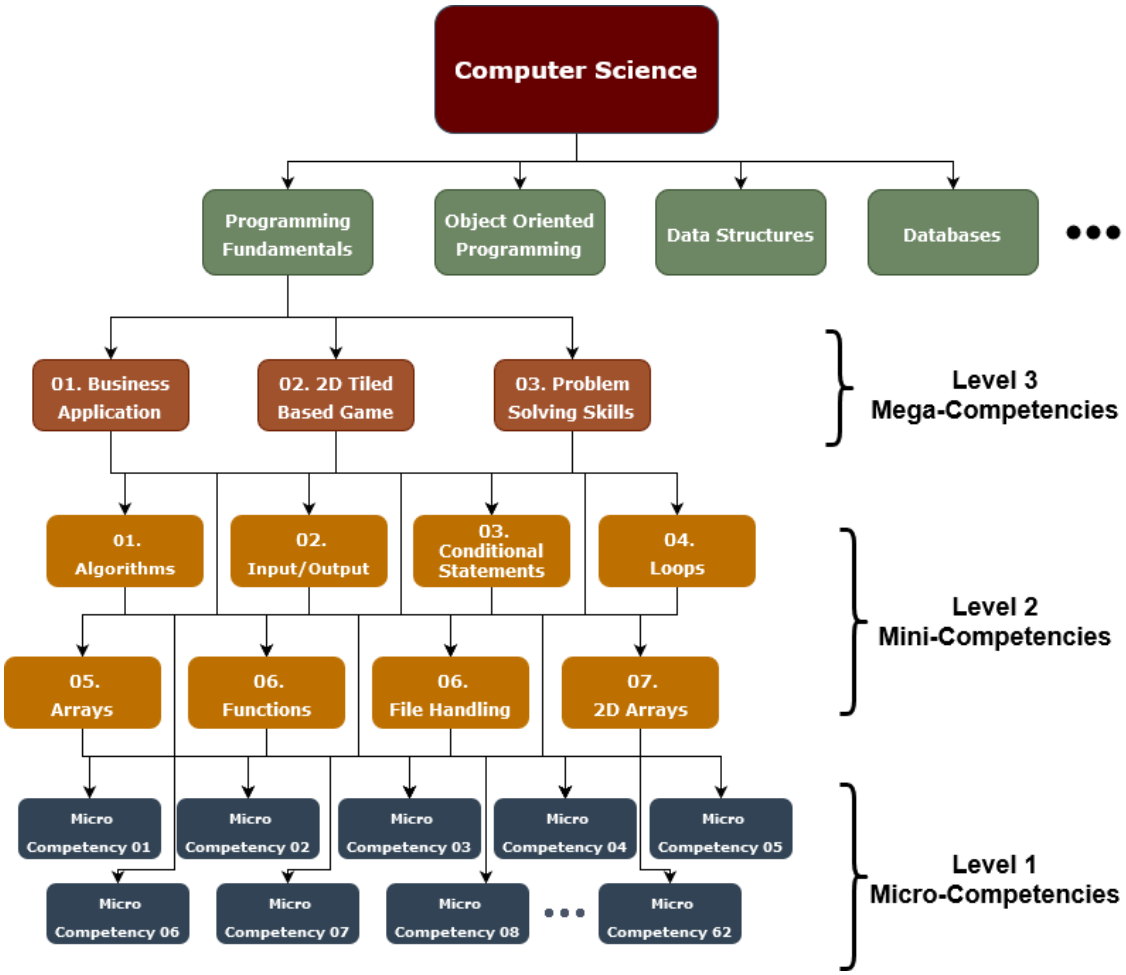
3. Research Methodology

The methodology of designing the Computing Curricula is composed of the following parts as shown in Figure 1.

- Mega Competencies (Level 3)
- Mini Competencies (Level 2)
- Micro Competencies (Level 1)

Figure 1

Methodology of Introductory programming course design based on Competency-based Learning



SFIA framework has provided 102 skills and competencies that the CS graduates require. These are distributed into multiple subjects during a four-year Computer Science degree program. According to our framework, each Computer Science course is divided into multiple mega competencies. These mega competencies are divided into multiple mini competencies that can be covered in about two weeks. These mini competencies are further divided into micro competencies that can be covered in a 2-3 hour lecture duration.

This research paper only focuses on defining the competencies of introductory programming courses.

Mega Competencies: Introductory Programming Course

Mostly, CS graduates are required to work on business applications or game development in their professional life. Therefore, the objective of a computer science degree must be to produce such skilled graduates that have the matching skills needed in the IT industry. Thus, this research paper defines the following three mega competencies for the programming fundamentals course by following the guidelines provided by Computing Curriculum 2020.

1 Develop a monolithic in-memory business application for the console while the requirements and design are given (or a well-guided tutorial is present).

2 Develop a 2D tile-based game for the console while requirements and design are given (or a well-guided tutorial is present).

3 Solve real-world problems using computational thinking and programming constructs.

These competencies represent actual Graphical User Interface (GUI) based business applications and game development that the IT industry values, but we have restrictions on developing the user-friendly applications due to prior knowledge of novice learners. Therefore, currently, the introductory programming course focuses on console-based applications.

These mega competencies have greater complexity, so these are decomposed into eight mini competencies.

Mini Competencies

Following are the eight mini competencies.

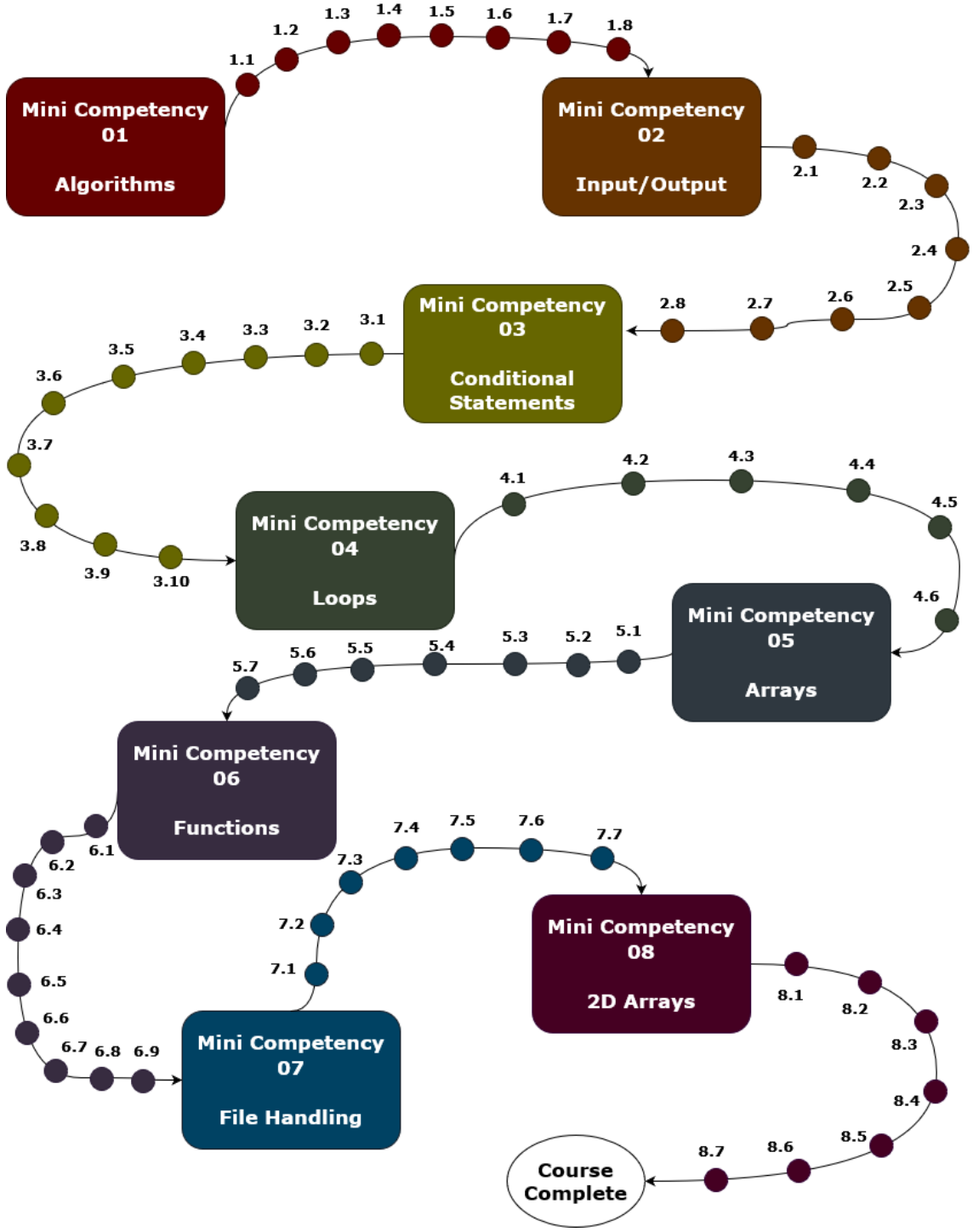
1. Analyze and Write Algorithms for converting one form of data (input) into another form of data using mathematical expressions with limited memory and available computational steps. (Algorithms)
2. Analyze and Write Computer Programs for converting one form of data (input) into another form of data using mathematical expressions. (Input/Output)
3. Analyze and Solve computational problems based on single and multiple conditions. (Conditional Statements)
4. Analyze and Solve complex computational problems involving repetition of dependent and independent iterations. (Loops)
5. Analyze and Solve complex computational problems involving large amounts of data. (Arrays)
6. Analyze and Solve complex computational problems by decomposing them into reusable blocks of code. (Functions)
7. Analyze and Solve complex computational problems by using persistently stored data. (File Handling)
8. Develop a 2D Game with the interaction of characters and a reward system (2D Arrays).

Every mini competency covers the basics of an introductory programming course (i.e., algorithms, input/output, conditional statements, loops, arrays, functions, file handling, and 2D arrays). It can be covered in a two-week duration. After completing the first seven mini-competencies, students can make any business application to store data into files, thus achieving the first mega competency. Mini competencies up to the file handling are the prerequisites of the second mega competency. Then the last mini competency is on 2D console-based game. Therefore, after completing all the eight mini competencies, students can easily develop any 2D tile-based game. The third mega competency is on problem-solving skills. Therefore, all mini competencies come under its umbrella.

The complete roadmap of the introductory programming course is given in Figure 2.

Figure 2

Roadmap of Introductory programming course



Micro Competencies

These mini competencies are divided into sixty-two micro competencies to teach at the lecture level. Skill levels of these micro-competencies are defined based on the levels of Bloom's taxonomy in the cognitive domain (Huitt, 2011). The contents to achieve these micro competencies are designed to engage the student in a thinking process instead of just delivering the concept to the student. These are designed by following the Four Component Instructional Design (4C/ID) model (Güney, 2019).

Each micro competency is drafted by dividing it into four categories.

1. Problem Visualization (Know-Why)
2. Knowledge Representation (Know-What)
3. Working Example (Know-How)
4. Self-Assessment (Reinforcement)

First, the students are shown the goal or vision of the lecture. A question then arises about how to achieve the goal. According to a study by Rosegard et al. (2013), raising questions is one of the attention-getting techniques and methods for classroom involvement. Furthermore, related questions are asked to ensure the student fully understands and grasps the problem at hand. He starts to think about what could be the possible solution and how to solve the problem or achieve the target. When the student is truly engaged, the instructor delivers the actual concept. The relevant knowledge required to solve the problem is provided to the student. Then, after delivering the knowledge, the instructor gives a working example to show how to use that knowledge to solve similar problems. Finally, students are given similar tasks in self-assessment to reinforce the learned concept.

The following are the micro competencies of the third mini competency (Conditional Statements)

1. Solve a problem using a single conditional statement with one Boolean expression consisting of an Equal comparison operator. (IF Statement)
2. Solve a problem using a single conditional statement with one Boolean expression consisting of any comparison operator. (IF Statement)
3. Solve a problem that involves multiple decision-making using conditional statements. (multiple IF Statements)
4. Solve a problem that makes multiple decision-making more optimized by checking fewer conditional statements. (IF-ELSE Statement)
5. Solve a problem that makes complex decision-making using nested conditional statements. (Nested IF Statement)
6. Solve a problem that involves complex decision-making using a single conditional statement with two Boolean expressions and the AND logical operator.
7. Solve a problem that involves complex decision-making using a single conditional statement with two Boolean expressions and the OR logical operator.
8. Solve a problem that involves complex decision-making using a single conditional statement with two Boolean expressions and the NOT logical operator.
9. Solve a problem that involves complex decision-making using a single conditional statement with multiple Boolean expressions and multiple logical operators.
10. Solve a problem that involves complex decision-making using a single conditional statement with multiple Boolean expressions and multiple logical operators while considering the precedence rules.

The complete Introductory programming course design, categorized into mega, mini, and micro competencies, is given in Appendix A.

4. Analysis and Results

Experimentation

The Introductory Programming course was taught in two sections (Section A and Section B) of the Computer Science program (Session 2020) at the undergraduate level at the University of Engineering and Technology, Lahore, Pakistan. The course was executed in three hours of theory lecture and nine hours of lab work per week.

Section A was taught the aforementioned designed Introductory Programming course based on the Competency-Based Learning (CBL) approach. In contrast, Section B was taught the traditional Introductory Programming course that followed the Knowledge-Based Learning (KBL) approach. Both sections had 50 students each. Section A contained 30 female and 20 male students, and Section B contained 26 female and 24 male students, as shown in Table 1.

Table 1

Students Data

Section	Methodology	Total Students	No. of Males	No. of Females
Section A	CBL	50	20	30
Section B	KBL	50	24	26

Both sections were judged on the same levels. They were taught the same knowledge elements (variables, arrays, loops, etc.), given the same assignments, quizzes, and Mid-term and final evaluation exams.

As the course progressed, the students of both sections were compared on the following two levels.

1. Affective Level
2. Cognitive Level

A survey checked the Affective level of the students after completing each micro-competency, i.e., after every two weeks (Appendix B). These surveys were given to the students after completing their assignment after every micro-competency, composed of real-life programming problems. This survey consisted of two parts.

1. How am I doing programming
2. How am I feeling

These parts were ranked from level 1 to 4, with level 4 being the highest, as shown in Table 2.

Table 2.

Motivation Levels based on programming and feeling levels of the student

Motivation Levels	How I am Doing Programming	How I am Feeling
Level 4	I did it without any help	**Confident!** It was a piece of cake.
Level 3	I did it with a little help.	**Thrilled!** It was challenging but I am feeling thrilled now.
Level 2	I made the logic but my code is not working.	**Confused!** I got confused after this exercise.
Level 1	I could not make the logic. I need more help to understand it	**Depressed!** I am depressed because I cannot do it.

The students ranked their progress and feeling level in their assignments, which defined the Affective level. The student's cognitive level was checked by quizzes, Mid-term, finals, business

application, and game project. There were two quizzes with 10 marks each; one before the Midterm and one before finals. The Midterm exam, consisting of 20 marks, was held after eight weeks of study, and the final exam, consisting of 40 marks, was held after a total of 16 weeks of study. A 10-mark Business Application project assessment was taken after completing seven mini-competencies. Game project assessment of 10 marks was taken after the completion of all eight mini competencies.

Table 3 shows the number of exams taken and their mark coverage.

Table 3.

Exams and their marks coverage

Exams	Marks
Quiz 1	10
Quiz 2	10
Business Application	10
Mid-term exam	20
2D Game	10
Finals	40

5. Results

This section presents the results of the experimentation mentioned above.

i. Affective level

Figure 3 and 4 shows the result of the survey conducted after covering mini-competency 3. The x-axis shows the feeling level and the y-axis shows the number of students on each level. Section A was taught by following the Competency based learning (CBL) and their results are represented by dark blue color. Section B was taught by following traditional Knowledge based learning (KBL) and their results are represented by light blue color.

Figure 3

Feeling level 1 and level 2 of the 2 Sections (Negative Feelings)

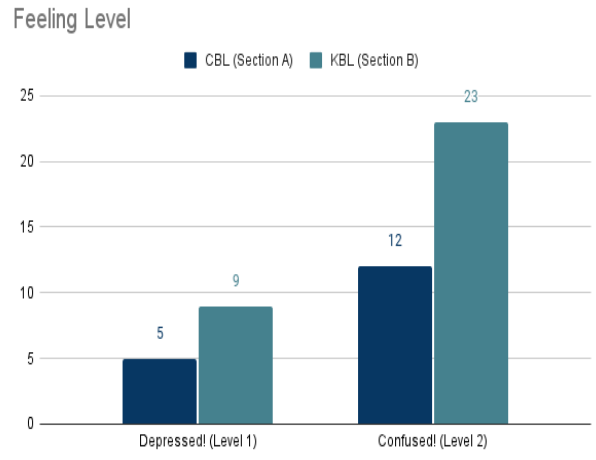


Figure 4

Feeling level 3 and level 4 of the 2 Sections (Positive Feelings)

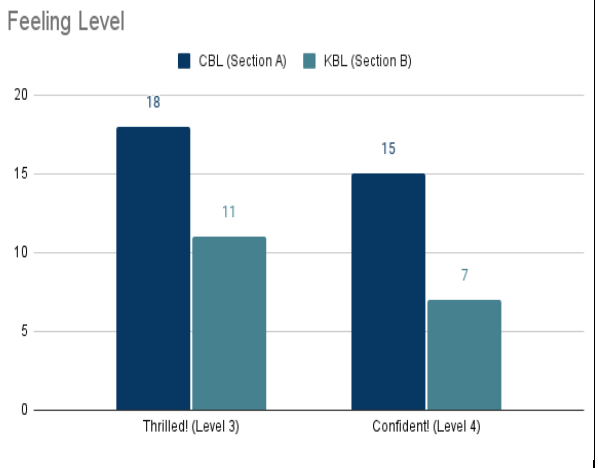


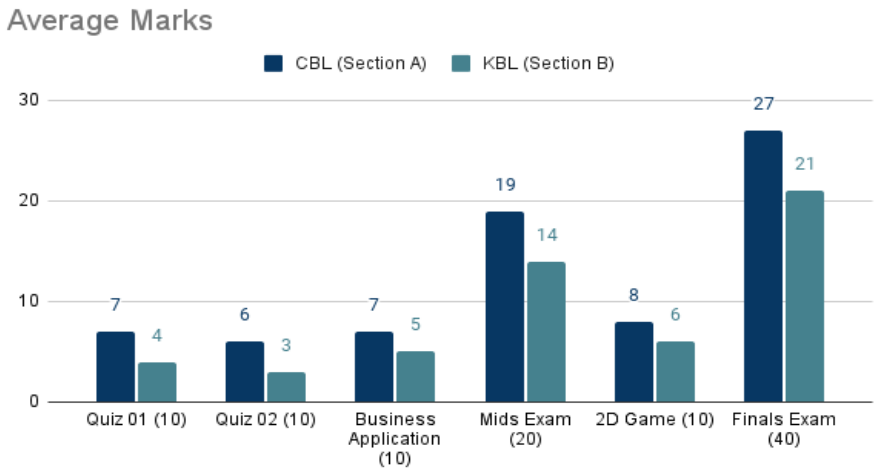
Figure 3 presents the number of responses of both sections on negative feelings, Level 1 (Depressed!) and Level 2 (Confused!), whereas Figure 4 presents the positive feelings, Level 3 (Thrilled!) and Level 4 (Confident!).

ii. Cognitive level

Figure 5 shows the average marks of quizzes, Mid-term, Final exam, business application, and game project of both sections separately. The x-axis shows the type of the exam, and the y-axis shows the average marks of 50 students. Section A was taught by following Competency-based learning (CBL), and their results are represented by dark blue color. Section B was taught by following the traditional Knowledge-based learning (KBL), and the results are represented by light blue color.

Figure 5

Average Exam marks of 2 Sections



6. Discussion

The major contribution of this research is of designing the mega, mini, and micro competencies of the introductory programming course based on 2020 computing curricula guidelines. Proposed mega competencies are carefully designed by matching the skills needed in the IT industry. These competencies give the bigger picture of what we want to accomplish that proves meaningful to both employers and students. Mega competencies are further divided into mini competencies, covering all the basics of introductory programming course and showing how to accomplish the mega competencies. After covering the first seven mini competencies, students can make any console-based business application. After covering the eighth mini competency, students can make any 2D console-based tile game. The mini competencies are further divided into micro competencies to form a concrete lecture plan to be easily executed in a classroom. Contents of the micro competencies are designed by following the 4CID model. Each micro competency starts with problem visualization. In this, students are anchored so that they fully understand the problem at hand. After that, students are only taught the concepts needed to solve the problem in the knowledge representation phase. Then they are presented with a working example and its solution to show how the concepts are used to solve the real-world problem. Then they are given a self-assessment at the end of the lecture to reinforce the concepts taught in the lecture.

When executed in the classroom, the designed course proved efficient in increasing the students' motivation, as shown in Figure 4. Section A students showed a more positive response

in their classes and assignments than Section B. Students of section A felt more confident and thrilled while studying the competency-based course than the students of section B studying the traditional course (Figure 4). This was because the section A students were shown the bigger picture, and they knew why they were studying the relevant concepts and why they needed to study those concepts. The programming tasks were given gradually with increasing complexity level after every micro competency and then more complex tasks after completing every mini competency. Therefore, Section A students were willing to do programming problems more by themselves because they could code themselves compared to the students of Section B. Section B students were more confused than section A students while doing the programming tasks (Figure 3). This was because the students of section B were bombarded with knowledge without motivating and inspiring them on why they were studying the related concepts. Section B students found it very difficult to apply the taught concepts in solving real-world problems. Thus, the overall Affective level of the students of section A was positive compared to the students of section B (Figure 3 and 4).

While covering the micro-competencies, students' skill levels increased more than the students of section B. Figure 5 shows the average marks of all the exams conducted. Section A performed better than section B in each and every exam. This was because the Section A students were internally motivated as they could solve the problems gradually with increasing complexity. Even the students of Section A came up with many innovative ideas for their business application and game project. They were thrilled to try and solve different and new problems by themselves, and they knew how the related concepts added up to complete the bigger picture.

Due to the Covid-19 situation, some of the classes had to be conducted online, but the overall response of the course designed based on competencies proved more efficient than the traditional introductory programming course.

7. Conclusion

After following the guidelines of ACM on computing curricula, we have designed an introductory programming course based on mega, mini, and micro competencies. The proposed competencies of the introductory programming course produce the programming skills in the students that the IT industry values. This course design provides a roadmap for instructors in lecture preparation to engage the students in the contents and keep their motivation intact. Results have proven that the students taught by following competency-based learning preferred to involve and solve programming problems actively than those taught by following the traditional knowledge-based learning.

To fulfil the need of the hour, our future work includes converting these competencies and micro-competencies course into an adaptive e-learning system. We also intend to develop the competencies and micro-competencies for object-oriented programming.

DECLARATION STATEMENTS

Conflict of Interest

The authors declare no actual or perceived conflicts of interest. They also confirm that no external funding was received for this study, beyond the allocation of academic time at their respective university.

Data Availability Statement

The data used in this study will be provided by the corresponding author on request.

Authors' Contribution

MUI: write-up, designing exams materials, Review, formatting. MS: Defining classroom competences, Methodology, Experimentation, Review and formatting. TW: Revie, logic structure, text re-articulation

8. References

Aspray, W. (2013). Computers, information, and everyday life. *IEEE Annals of the History of Computing*, *35*(4), 96. DOI: 10.1109/MAHC.2013.46

Barak, M., Harward, J., Kocur, G., & Lerman, S. (2007). Transforming an introductory programming course: From lectures to active learning via wireless laptops. *Journal of Science Education and Technology*, *16*(4), 325–336. DOI: 10.1007/s10956-007-9055-5

Begel, A., & Simon, B. (2008). Struggles of new college graduates in their first software development job. *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, 226–230. DOI: 10.1145/1352322.1352218

Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. *AcM SIGcSE Bulletin*, *39*(2), 32–36. DOI: 10.1145/1272848.1272879

Brown, J. (2020). An examination of the Skills Framework for the Information Age (SFIA) version 7. *International Journal of Information Management*, *51*, 102058. DOI: 10.1016/j.ijinfomgt.2019.102058

Carnevale, A. P., & Smith, N. (2013). Workplace basics: The skills employees need and employers want. In *Human Resource Development International* (Vol. 16, Issue 5, pp. 491–501). Taylor & Francis. DOI: 10.1080/13678868.2013.821267

Cassel, L., Clements, A., Davies, G., Guzdial, M., McCauley, R., McGettrick, A., Sloan, B., Snyder, L., Tymann, P., & Weide, B. W. (2008). *Computer science curriculum 2008: An interim revision of CS 2001*. ACM. DOI: https://doi.org/10.1145/2593246

Cheah, C. S. (2020). Factors contributing to the difficulties in teaching and learning of computer programming: A literature review. *Contemporary Educational Technology*, *12*(2), ep272. DOI: 10.30935/cedtech/8247

Clear, A., Parrish, A. S., Impagliazzo, J., & Zhang, M. (2019). Computing Curricula 2020: introduction and community engagement. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 653–654. DOI: https://doi.org/10.1145/3287324.3287517

Dagiene, V., Hromkovic, J., & Lacher, R. (2021). Designing informatics curriculum for K-12 education: From Concepts to Implementations. *Informatics in Education*, *20*(3), 333–360. DOI: 10.15388/infedu.2021.22

Draft, S. (2013). Computer Science Curricula 2013. *ACM and IEEE Computer Society, Incorporated: New York, NY, USA*.

https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=e32b65428b8ea54fd628839fd7386c88716a319b

Dziallas, S., & Fincher, S. (2015). ACM Curriculum Reports: A pedagogic perspective. Proceedings of the Eleventh Annual International Conference on International Computing Education Research, 81–89. https://doi.org/10.1145/2787622.2787714

Force, C. T. (2020). Computing Curricula 2020: Paradigms for Global Computing Education November 2020, ACM, NY, US. https://doi.org/10.1145/3467967

Gavrilović, N., Arsić, A., Domazet, D., & Mishra, A. (2018). Algorithm for adaptive learning process and improving learners' skills in Java programming language. *Computer Applications in Engineering Education*, 26(5), 1362–1382. DOI: 10.1002/cae.22043

Geissler, M., Brown, D., McKenzie, N., Peltsverger, S., Preuss, T., Sabin, M., & Tang, C. (2020). Information Technology Transfer Curricula 2020: Curriculum Guidelines for Two-Year Transfer Programs in Information Technology. ACM. https://dx.doi.org/10.1145/3414584

Güney, Z. (2019). Four-Component Instructional Design (4C/ID) Model Approach for Teaching Programming Skills. International Journal of Progressive Education, 15(4), 142–156. DOI: 10.29329/ijpe.2019.203.11

Hannay, J. E., Dybå, T., Arisholm, E., & Sjøberg, D. I. K. (2009). The effectiveness of pair programming: A meta-analysis. Information and Software Technology, 51(7), 1110–1122. https://doi.org/10.1016/j.infsof.2009.02.001

Harvard University Competency Dictionary FY14 (2014), Harvard University Website http://hms.harvard.edu/, 2014.

Havenga, H. M. (2015). Project-based learning in higher education: exploring programming students' development towards self-directedness. South African Journal of Higher Education, 29(4), 135–157. https://hdl.handle.net/10520/EJC182452

Hennig-Thurau, T., Ravid, S. A., & Sorenson, O. (2021). The economics of filmed entertainment in the digital era. In Journal of Cultural Economics (Vol. 45, Issue 2, pp. 157–170). Springer. 10.1007/s10824-021-09407-6

Hetling, A., Watson, S., & Horgan, M. (2014). "We live in a technological era, whether you like it or not" client perspectives and online welfare applications. Administration & Society, 46(5), 519–547. 10.1177/0095399712465596

Huitt, W. (2011). Bloom et al.'s taxonomy of the cognitive domain. Educational Psychology Interactive, 22. http://www.edpsycinteractive.org/topics/cognition/bloom.html

Kaharuddin, A. (2020). Communicative Competence-Based Syllabus Design for Initial English Speaking Skills. Available at SSRN: https://ssrn.com/abstract=3559105.

Kim, Y., Hsu, J., & Stern, M. (2006). An update on the IS/IT skills gap. Journal of Information Systems Education, 17(4), 395. https://jise.org/volume17/n4/JISEv17n4p395.html

Labor, U. D. (2017). Occupational Outlook Handbook (OOH), US Bureau of Labor Statistics. Retrieved From. https://www.bls.gov/ooh/computer-and-information-technology/home.htm

Liu, J., & He, L. (2015). Practical skills training in computer education. International Journal of Information & Computer Science, 4, 25–29. https://doi.org/10.14355/IJICS.2015.04.004

Longenecker, B., Babb, J. S., Waguespack, L., Janicki, T., & Feinstein, D. (2015). Establishing the Basis for a CIS (Computer Information Systems) Undergraduate Degree Program: On Seeking the Body of Knowledge. Information Systems Education Journal, 13(5), 37. https://isedj.org/2015-13/n5/ISEDJv13n5p37.html

Marcus, J. (2017). Competency-based education, put to the test: An inside look at Learning and Assessment at Western Governors University, Education Next, 17(4), 27–33, Fall 2017.

McGlochlin, T. (2018). 5 Stats About the Skills Gap that Demand Attention, PSI Select International, 2018. https://blog.psionline.com/talent/5-stats-about-the-skills-gap-7684 that-demands-attention

Petersen, A., Craig, M., Campbell, J., & Tafliovich, A. (2016). Revisiting why students drop

CS1. *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, 71–80. https://doi.org/10.1145/2999541.2999552

Rojas-López, A., & Garcia-Peñalvo, F. J. (2019). Initial learning scenarios based on the computational thinking evaluation for the course Programming fundamentals at INACAP. *Proceedings of the Seventh International Conference on Technological Ecosystems for Enhancing Multiculturality*, 6–12. https://doi.org/10.1145/3362789.3362802

Rosegard, E., & Wilson, J. (2013). Capturing students' attention: An empirical student. *Journal of the Scholarship of Teaching and Learning*, 1–20. https://scholarworks.iu.edu/journals/index.php/josotl/article/view/3891

Shackelford, R., McGettrick, A., Sloan, R., Topi, H., Davies, G., Kamali, R., Cross, J., Impagliazzo, J., LeBlanc, R., & Lunt, B. (2006). Computing curricula 2005: The overview report. *ACM SIGCSE Bulletin*, *38*(1), 456–457. https://doi.org/10.1145/1124706.1121482

Sobral, S. R. (2021). Strategies on teaching introducing to programming in higher education. *World Conference on Information Systems and Technologies*, 133–150. DOI: 10.1007/978-3-030-72660-7_14

Souza, M., Rodrigues, P. (2015). Investigating the effectiveness of the flipped classroom in an introductory programming course. *The New Educational Review*, *40*(1), 129–139. DOI: 10.15804/tner.2015.40.2.11

Sultana, S. (2016). *Defining the competencies, programming languages, and assessments for an introductory computer science course*. Old Dominion University. DOI: 10.25777/sgra-pa16

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

von Konsky, B. R., Miller, C., & Jones, A. (2016). The skills framework for the information age: Engaging stakeholders in curriculum design. *Journal of Information Systems Education*, *27*(1), 37. https://jise.org/volume27/n1/JISEv27n1p37.html

Voskoglou, M. (2021). Computers and Artificial Intelligence in Future Education. In *Handbook of Research on Teaching With Virtual Environments and AI* (pp. 654–680). IGI Global. DOI: 10.4018/978-1-7998-7638-0.ch028

Watson, C., & Li, F. W. B. (2014). Failure rates in introductory programming revisited. *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, 39–44. https://doi.org/10.1145/2591708.2591749

Whitehall, B. L., & Lu, S. C. Y. (1994). Theory completion using knowledge-based learning. *Machine Learning: A Multistrategy Approach*, 165.

Woodley, M., & Kamin, S. N. (2007). Programming studio: A course for improving programming skills in undergraduates. *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, 531–535. https://doi.org/10.1145/1227310.1227490

Zhang, L., Nouri, J., & Rolandsson, L. (2020). Progression of Computational Thinking skills in Swedish compulsory schools with block-based programming. *Proceedings of the Twenty-Second Australasian Computing Education Conference*, 66–75. https://doi.org/10.1145/3373165.3373173

Appendix A

Design of the competencies based on the guidelines given in ACM Curriculum 2020.

Competency Title: Console Based Business Application		
Mega Competency 01 Develop a monolithic in-memory console-based business application for admin, employers and customers while the requirements and design are provided		
Mini Competency 01 Analyze and Write Algorithms for converting one form of data (input) into another form of data using mathematical expressions with limited memory and available computational steps (Algorithms)		
Micro Competencies	**Knowledge Elements**	**Bloom's Taxonomy Skill Level**
Explain the Hardware Components (IO Devices, CPU, memory) and their Role while Computing any Problem.	1. Architecture and Organization • IPO Cycle, Memory, CPU, Fetch Decode Cycle	Understand
Explain the Instruction Code, Computational Steps and Algorithm	1 Architecture and Organization • IPO Cycle, Memory, CPU, Fetch Decode Cycle • Programming Fundamentals • Algorithms	Understand
Define the Binary Language and its Relationship with Algorithm and Program	1. Programming Fundamentals • Algorithms • Program	Understand
Explain Machine Language, High Level Languages and Role of Compiler	1. Programming Fundamentals • Algorithms • Program	Understand
Define what is Computational Thinking and its four pillars	1. Analytical and Critical Thinking	Understand
Write an Algorithm that converts Input into the required Output using Variables, Constants and Arithmetic Operators.	1. Programming Fundamentals • Variables • Expressions • Arithmetic operators	Apply
Evaluate the Correctness of Algorithms with Different Test Cases.	1. Programming Fundamentals • Algorithms • Testing and debugging	Understand
Mini Competency 02 Analyze and Write Computer Programs for converting one form of data (input) into another form of data using mathematical expressions (Input/Output)		
Micro Competencies	**Knowledge Elements**	**Bloom's Taxonomy Skill Level**
Write and execute a computer program that shows output on monitor screen (Console)	1. Programming Fundamentals • Output on Console	Apply

Explain why we need Variables and what is their Relation with the Memory.	1. Programming Fundamentals • Variables 2. Architecture and Organization • Memory	Remember
Explain what is a Data Type, why we need it and what is its Role in Variable Declaration	1. Programming Fundamentals • Variables • Datatypes	Remember
Write expressions using Variables, Constants and Arithmetic Operators	1. Programming Fundamentals • Variables • Expressions • Arithmetic operators	Apply
Write Expressions while considering the Operator Precedence Rules	1. Programming Fundamentals • Variables • Expressions • Arithmetic operators • Precedence Rule	Apply
Write a Program in C++ that Declares Variable, Stores Value in it and Prints its Value on Console	1. Programming Fundamentals • Variables • Output on Console	Apply
Write a C++ program that evaluates expressions consisting of Arithmetic Operators, Constants and Variables	1. Programming Fundamentals • Variables • Expressions • Arithmetic operators • Output on Console	Apply
Write a C++ program that takes input from the user, applies mathematical operations on it and then converts that input into output.	1. Programming Fundamentals • Input • Variables • Expressions • Arithmetic operators • Output on Console	Apply
Mini Competency 03 Analyze and Solve computational problems based on single and multiple conditions (Conditional Statements).		
Micro Competencies	**Knowledge Elements**	**Bloom's Taxonomy Skill Level**
Write a C++ program using a single conditional statement with one Boolean expression consisting of an Equal comparison operator. (IF Statement)	1. Programming Fundamentals • IF statement • Comparison Operators	Apply
Write a C++ program using a single conditional statement with one Boolean expression consisting of Any comparison operator. (IF Statement)	1. Programming Fundamentals • IF statement • Comparison Operators	Apply
Write a C++ program that involves multiple decision making using conditional statements. (Multiple IF Statements)	1. Programming Fundamentals • Multiple IF statements • Comparison Operators	Apply

Write a C++ program that makes multiple decision making more optimized by checking fewer conditional statements. (IF-Else Statement)	1. Programming Fundamentals • IF-Else statements • Comparison Operators	Apply
Write a C++ program that makes complex decision making using nested conditional statements. (Nested IF Statement)	1. Programming Fundamentals • Nested IF statements • Comparison Operators	Apply
Write a C++ program that involves complex decision making using a single conditional statement with two Boolean expressions and the AND logical operator.	1. Programming Fundamentals • Conditional statements • Comparison Operators • Logical Operators	Apply
Write a C++ program that involves complex decision making using a single conditional statement with two Boolean expressions and the OR logical operator.	1. Programming Fundamentals • Conditional statements • Comparison Operators • Logical Operators	Apply
Write a C++ program that involves complex decision making using a single conditional statement with two Boolean expressions and the NOT logical operator.	1. Programming Fundamentals • Conditional statements • Comparison Operators • Logical Operators	Apply
Write a C++ program that involves complex decision making using a single conditional statement with multiple Boolean expressions and multiple logical operators.	1. Programming Fundamentals • Conditional statements • Comparison Operators • Logical Operators	Apply
Write a C++ program that involves complex decision making using a single conditional statement with multiple Boolean expressions and multiple logical operators while considering the precedence rules.	1. Programming Fundamentals • Conditional statements • Comparison Operators • Logical Operators • Precedence Rules	Apply
Mini Competency 04 Analyze and Solve complex computational problems involving repetition of dependent and independent iterations (Loops)		
Micro Competencies	**Knowledge Elements**	**Bloom's Taxonomy Skill Level**
Write a C++ Program that repeats a Set of Instructions for a specific number of times to solve the given problem using Counter Loop.	1. Programming Fundamentals • Counter Loop (For Loop)	Apply
Write a C++ Program that repeats a Set of Instructions for an unknown number of times to solve the given problem using Conditional Loop.	1. Programming Fundamentals • Conditional Loop (While Loop)	Apply
Write a C++ Program that repeats a Set of Instructions for any number of times to solve the given problem using Counter Loop.	1. Programming Fundamentals • Counter Loop (For Loop)	Apply

Write a C++ Program that solves larger problems by decomposing it into smaller subproblems and combining their solution to achieve final results using the Counter Loop.	1. Programming Fundamentals • Computational thinking • Counter Loop (For Loop)	Apply
Write a C++ Program that repeats a complex set of instructions using nested loops.	1. Programming Fundamentals • Counter Loop (For Loop) • Nested Loops	Apply
Write a program that alters the normal flow of the loop using continue and break statements.	1. Programming Fundamentals • Counter Loop (For Loop) • Continue Statement • Break Statement	Apply
Mini Competency 05 Analyze and Solve complex computational problems involving large amount of data (Arrays).		
Micro Competencies	**Knowledge Elements**	**Bloom's Taxonomy Skill Level**
Identify the scenarios when scalar variables are not sufficient to handle the data and subsequently arrays are required.	1. Programming Fundamentals • Limitation of variables	Analyze
Write a C++ program that declares an Array, takes a large number of elements as input from the user in that array and then retrieves that data from the array	1. Programming Fundamentals • Arrays • CRUD operations on Arrays	Apply
Write a C++ program that declares an Array, takes an unknown number of elements as input from the user and then retrieves that data from the array	1. Programming Fundamentals • Arrays • CRUD operations on Arrays	Apply
Write a C++ program that processes data elements in the array and make them in an order.	1. Programming Fundamentals • Arrays • CRUD operations on Arrays	Apply
Write a C++ program for storing and processing multiple information of a record using parallel Arrays.	1. Programming Fundamentals • Parallel Arrays • CRUD operations on Parallel Arrays	Apply
Mini Competency 06 Analyze and Solve complex computational problems by decomposing into reusable blocks of code (Functions)		
Micro Competencies	**Knowledge Elements**	**Bloom's Taxonomy Skill Level**
Write a Code that requires lengthy and repeated structure to solve complex problems.	1. Programming Fundamentals 2. Analytical and Critical Thinking	Apply
Identify the problems that arise due to the lengthy and repeated code for complex problems	1. Programming Fundamentals 2. Analytical and Critical Thinking	Analyze

Write a program that calls predefined functions to generate the results.	1. Programming Fundamentals • Functions (Pre-defined)	Apply
Write a program that calls a user-defined function to generate the results	1. Programming Fundamentals • Functions (User-defined)	Apply
Write a user defined function to solve the code repetition issue.	1. Programming Fundamentals • Functions (User-defined) 2. Analytical and Critical Thinking	Apply
Understand the importance of function prototype.	1. Programming Fundamentals • Functions (User-defined) • Function Prototype	Remember
Write a code that uses global variables for communicating between functions.	1. Programming Fundamentals • Functions (User-defined) • Function Prototype • Global variables	Apply
Identify the difference between High coupled and Low coupled functions.	1. Programming Fundamentals • Functions (User-defined) • High VS Low coupled Functions	Analyze
Write a highly cohesive and low coupled function that can be reused in complex problems.	1. Programming Fundamentals • Functions (User-defined) • Function Prototype • Global variables 2. Analytical and Critical Thinking	Apply
Mini Competency 07 Analyze and Solve complex computational problems by using persistently stored data (File Handling)		
Micro Competencies	**Knowledge Elements**	**Bloom's Taxonomy Skill Level**
Write a Code to show the limitations of taking input and then displaying the output on the console	1. Programming Fundamentals • File Handling 2. Analytical and Critical Thinking	Apply
Write a code that solves the limitations of taking input from the user	1. Programming Fundamentals • File Handling 2. Analytical and Critical Thinking	Apply
Write a code that reads character by character from the file.	1. Programming Fundamentals • File Handling • CRUD operations using files	Apply
Write a code that creates (store) data into the permanent storage	1. Programming Fundamentals • File Handling • CRUD operations using files	Apply
Write a code that appends (insert) data into the permanent storage.	1. Programming Fundamentals • File Handling • CRUD operations using files	Apply
Write a code that searches from the formatted (comma separated file) storage.	1. Programming Fundamentals • File Handling • CRUD operations using files	Apply

Development of data driven applications with permanent storage.	1. Programming Fundamentals • File Handling 2. Analytical and Critical Thinking	Create
Competency Title: Console based tiled 2D Game		
Mega Competency 02 Develop a monolithic in-memory console-based 2D Tile Game while the requirements and design are provided		
Prerequisite: Mini Competency 01 to Mini Competency 07		
Mini Competency 08 Develop a 2D Game with interaction of characters and reward system		
Micro Competencies	**Knowledge Elements**	**Bloom's Taxonomy Skill Level**
Write a program that stores similar records using two dimensional arrays.	1. Programming Fundamentals • 2D Arrays	Apply
Write a story for the 2D tile-based game, identify the characters (Player and enemies), game rules, goals, interactions and reward system	1. Analytical and Critical Thinking	Remember
Write a program that stores game maps and game objects into the 2D Array and show the map with the objects on the console from the 2D Array	1. Programming Fundamentals • 2D Arrays	Apply
Write a program to move a game object on the console using arrow keys.	1. Programming Fundamentals • Key Strokes 2. Analytical and Critical Thinking	Apply
Write a program to detect interaction between objects and update the game state accordingly.	1. Programming Fundamentals • 2D Arrays 2. Analytical and Critical Thinking	Apply
Write a program that stores data of a 2D array into file and loads the data back to 2D Array using file handling.	1. Programming Fundamentals • 2D Arrays • File Handling 2. Analytical and Critical Thinking	Apply
Write a program that saves the Game Map into the file and loads it back to the 2D array.	1. Programming Fundamentals • 2D Arrays • File Handling 2. Analytical and Critical Thinking	Apply

Develop the Pac-Man game by loading data from the file.	1. Programming Fundamentals • 2D Arrays • File Handling 2. Analytical and Critical Thinking	Create

Competency Title: Problem Solving
Mega Competency 03 Solve real-world problems using computational thinking and programming constructs.
Prerequisite: Mini Competency 01 to Mini Competency 08

Appendix B

Sample self-assessment programming problems of Micro Competency 03 are as follows:

Problem 1

A hotel offers **two types of rooms**: **studio and apartment**. Prices are in dollars ($). Write a program that calculates **the price of the whole stay for a studio and apartment**. **Prices** depend on the **month** of the stay:

May and October	June and September	July and August
Studio- 50$ / per night	Studio - 75$ / per night	Studio 76$ / per night
Apartment - 65$ / per night	Apartment - 68$ /per night	Apartment - 77$ / per night

The following **discounts** are also offered:

- For a **studio**, in case of **more than 7** stays in **May and October**: **5% discount**.
- For a **studio**, in case of **more than 14** stays in **May and October**: **30% discount**.
- For a **studio**, in case of **more than 14** stays in **June and September**: **20% discount**.
- For an **apartment**, in case of **more than 14** stays, **no limitation regarding the month: 10% discount**.

Input Data

The input data is read from the **console** and contains **exactly two lines**:

- The **first** line contains the **month** – **May**, **June**, **July**, **August**, **September** or **October**.
- The **second** line is the **number of stays** – **integer within the range** [**0 … 200**].

Output Data

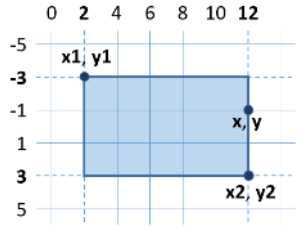
Print the following **two lines** on the console:

- On the **first line**: "**Apartment: { price for the whole stay }$.**"
- On the **second line**: "**Studio: { price for the whole stay }$.**"

Input	Output	Comments
May 15	Apartment: 877.50$. Studio: 525.00$.	In May, in the case of more than 14 stays, the discount for the studio is 30% (50 - 15 = 35), and for the apartment is 10% (65 - 6.5 = 68.5). The whole stay in the apartment: 877.50 lv The whole stay in the studio: 525.00 lv
June 14	Apartment: 961.80$. Studio: 1052.80$.	
August 20	Apartment: 1386.00$. Studio: 1520.00$.	

Problem 2

Write a program that checks whether a **point {x, y}** is placed **onto any of the sides of a rectangle {x1, y1} – {x2, y2}**. The input data is read from the console and consists of 6 lines: the decimal numbers **x1**, **y1**, **x2**, **y2**, **x** and **y** (as it is guaranteed that **x1 < x2** and **y1 < y2**). Print "**Border**" (if the point lies on any of the sides) or "**Inside / Outside**" (in the opposite case).



Test Cases:

Input	Output
2 -3 12 3 12 -1	Border
2 -3 12 3 8 -1	Inside / Outside

Problem 3

A student has to attend **an exam at a particular time** (for example, at 9:30 am). They arrive in the exam room at a particular **time of arrival** (for example 9:40 am). It is considered that the student has arrived **on time** if they have arrived **at the time when the exam starts or up to half an hour earlier**. If the student has arrived **more than 30 minutes earlier**, the student has come **too early**. If they have arrived **after the time when the exam starts**, they are **late**.

Write a program that inputs the exam starting time and the time of student's arrival, and prints if the student has arrived **on time**, if they have arrived **early** or if they are **late**, as well as **how many hours or minutes** the student is early or late.

Input Data

Read the following **four integers** (one on each line) from the console:

- The first line contains **exam starting time (hours)** – an integer from 0 to 23.
- The second line contains **exam starting time (minutes)** – an integer from 0 to 59.
- The third line contains an hour **of arrival** – an integer from 0 to 23.
- The fourth line contains **minutes of arrival** – an integer from 0 to 59.

Output Data

Print the following on the first line on the console:

- "**Late**", if the student arrives **later** compared to the exam starting time.
- "**On time**", if the student arrives **exactly** at the exam starting time or up to 30 minutes earlier.
- "**Early**", if the student arrives more than 30 minutes **before** the exam's starting time.

If the student arrives with more than one minute difference compared to the exam starting time, print on the next line:

- "**mm minutes before the start**" for arriving less than an hour earlier.
- "**hh:mm hours before the start**" for arriving 1 hour or earlier. Always print minutes using 2 digits, for example "1:05".

- "**mm minutes after the start**" for arriving more than an hour late.
- "**hh:mm hours after the start**" for arriving late by 1 hour or more. Always print minutes using 2 digits, for example, "1:03".

Test Cases

Input	Output
Exam Starting Time (hour): 9 Exam Starting Time (minutes): 30 Student hour of arrival: 9 Student minutes of arrival: 50	Late 20 minutes after the start
Exam Starting Time (hour): 16 Exam Starting Time (minutes): 0 Student hour of arrival: 15 Student minutes of arrival: 0	Early 1:0 hours before the start
Exam Starting Time (hour): 9 Exam Starting Time (minutes): 0 Student hour of arrival: 8 Student minutes of arrival: 30	On time 30 minutes before the start

Problem 4

Vladimir is a student, lives in Sofia, and goes to his hometown from time to time. He is very keen on volleyball, but is busy during weekdays and plays **volleyball** only during **weekends** and on **holidays**. Vladimir plays **in Sofia** every **Saturday**, when **he is not working**, and **he is not travelling to his hometown** and also during **2/3 of the holidays**. He travels to his **hometown h times** a year, where he plays volleyball with his old friends on **Sunday**. Vladimir **is not working 3/4 of the weekends**, during which he is in Sofia. Furthermore, during **leap years,** Vladimir plays **15% more** volleyball than usual. We accept that the year has exactly **48 weekends**, suitable for volleyball. Write a program that calculates **how many times Vladimir has played volleyball** throughout the year. **Round the result** down to the nearest whole number (e.g. 2.15 -> 2; 9.95 -> 9).

The input data is read from the console:

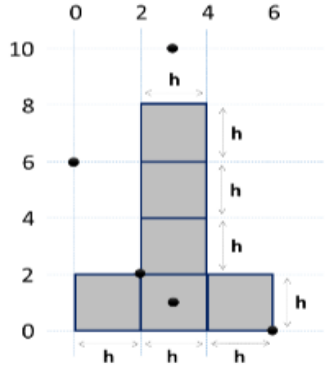
- The first line contains the word "**leap**" (leap year) or "**normal**" (a normal year with 365 days).
- The second line contains the integer **p** – the count of holidays in the year (which are not Saturday or Sunday).
- The third line contains the integer **h** – the count of weekends, in which Vladimir travels to his hometown.

Test Cases:

Input	Output
leap 5 2	45
normal 3 2	38
normal 11 6	44
leap 0 1	41

Problem 5

1. The figure consists of **6 blocks with size h * h**, placed as in the figure below. The lower left angle of the building is at position {0, 0}. The upper right angle of the figure is on position {**2*h**, **4*h**}. The coordinates given in the figure are for **h = 2**



Write a program that enters an integer **h** and the coordinates of a given **point {x, y}** (integers) and prints whether the point is inside the figure (**inside**), outside of the figure (**outside**) or on any of the borders of the figure (**border**).

Test Cases:

Input	Output
2 3 10	Outside
2 2 2	Border
15 13 55	Outside
15 29 37	Inside

Problem 6

It is strange, but most people start planning their vacations well in advance. A young programmer from New York has a certain **budget** and spare time in a particular **season**.
Write a program that accepts **as input the budget** and **season**, and **as output** displays the programmer's **vacation place** and **the amount of money they will spend**.
The budget determines the destination, and the season determines what amount of the budget will be spent. If the season is **summer**, the programmer will go **camping**, if it is **winter – they will stay in a hotel**. If it is in **Europe**, **regardless of the season**, the programmer will stay in a **hotel**. Each **camp** or **hotel**, **according to the destination**, has **its own price**, which corresponds to a particular **percentage of the budget**:

- If **100$ or less** – somewhere in **Bulgaria**.
  - **Summer** – **30%** of the budget.
  - **Winter** – **70%** of the budget.
- If **1000$ or less** – somewhere in the **Balkans**.
  - **Summer** – **40%** of the budget.
  - **Winter** – **80%** of the budget.
- If **more than 1000$** – somewhere in **Europe**.
  - Upon travelling in Europe, regardless of the season, the programmer will spend **90% of the budget**.

Input Data
The input data will be read from the console and will consist of **two lines**:

- The **first** line holds **the budget** – **real number** in the range [**10.00 … 5000.00**].
- The **second** line holds **one** of two possible seasons: "**summer**" or "**winter**".

Output Data
Two lines must be printed on the console.

- On the **first** line – "**Somewhere in {destination}**" among "**Bulgaria**", "**Balkans**" and "**Europe**".
- On the **second** line – "{**Vacation type**} – {**Amount spent**}".
  - The **Vacation** can be in a "**Camp**" or "**Hotel**".

Test Cases:

Input	Output
50 summer	Somewhere in Bulgaria Camp - 15.00
75 winter	Somewhere in Bulgaria Hotel - 52.50
312 summer	Somewhere in Balkans Camp - 124.80
678.53 winter	Somewhere in Balkans Hotel - 542.82
1500 summer	Somewhere in Europe Hotel - 1350.00

Sample Survey that was filled after completing the self-assessments by each student.

Tasks	How I am Doing Programming				How I am Feeling			
	Level 4	**Level 3**	**Level 2**	**Level 1**	**Level 4**	**Level 3**	**Level 2**	**Level 1**
	I did it without any help	I did it with a little help.	I made the logic but my code is not working	I could not make the logic. I need more help to understand it	**Confident!** It was a piece of cake.	**Thrilled!** It was challenging but I am feeling thrilled now.	**Confused!** I got confused after this exercise.	**Depressed!** I am depressed because I cannot do it.
MC 01								
MC 02								
MC 03								
MC 04								
MC 05								
MC 06								
MC 07								
MC 08								