# A genetic algorithm-based approach for making pairs and assigning exercises in a programming course

**Article** *in* Computer Applications in Engineering Education · October 2020

CITATIONS
0

READS
94

**3 authors**, including:

Kanika Kanika
Netaji Subhas Institute of Technology
**10** PUBLICATIONS **42** CITATIONS

SEE PROFILE

Received: 16 March 2020 | Accepted: 26 September 2020

DOI: 10.1002/cae.22349

RESEARCH ARTICLE

WILEY

# A genetic algorithm-based approach for making pairs and assigning exercises in a programming course

**Kanika Tehlan** | **Shampa Chakraverty** | **Pinaki Chakraborty** | **Shraddha Khapra**

Department of Computer Engineering, Netaji Subhas Institute of Technology, Delhi, India

**Correspondence**
Kanika Tehlan, Department of Computer Engineering, Netaji Subhas Institute of Technology, Delhi 110078, India.
Email: kanikatehlan@gmail.com

**Abstract**

Pair programming is an approach where two programmers work to solve one programming problem sitting shoulder to shoulder on a computer. Several studies indicating numerous benefits of using pair programming as a teaching strategy exist. However, only a few of them take into consideration the mechanism followed for pair formation. With an aim to study the impact of pair programming on undergraduate students, we try to make the pairs compatible with a genetic algorithm-based approach. Using a genetic algorithm, the system ensures that every pair in the class gets a particular combination of skills and personality traits. We also developed a desktop application to assign programming exercises to students dynamically. To assess the efficacy of pair programming in introductory programming course, a formal pair programming experiment was run at Netaji Subhas University of Technology. The pair programming experiment involved a total 171 undergraduate students from a computer engineering course. At the end of the program, we assessed the programming abilities of every student. We also analyzed the impact of a genetic algorithm-based pairing mechanism. On the basis of assessments, it is observed that pair programming is a successful pedagogical tool for facilitating active learning of introductory programming courses. Responses to survey garnered from undergraduate students hint that the genetic algorithm approach leads to compatible pairs.

**KEYWORDS**

collaborative learning, genetic algorithm, introductory programming, pair programming

## 1 | INTRODUCTION

Traditionally, students were made to program on separate computers. However, during the last few decades, pair programming is one of the several approaches used to make introductory programming easier and interesting for students [24,27]. Pair programming originates from the extreme programming methodology [3], a programming methodology extensively used in the industry. The basic idea behind pair programming is that knowledge is constructed in a social environment through collaborative efforts toward shared objectives [36]. In collaborative learning, pair programming is a practice where two students exchange thoughts and work to solve programming problems together [12]. Rather than a single student struggling to get the code and logic right, two brains perform analysis, design, and coding together [3]. While programming in pairs, there is a continuous exchange of

  

knowledge between the partners [6]. Two individuals hence work to achieve the same aim, share resources, and also share knowledge. This makes solving programming problems easier and faster [40]. Various benefits of pair programming have been reported so far. Technical skills, communication, and interpersonal skills improve while programming in pairs [4]. Continuous discussion and constant code reviews help the students produce code with fewer mistakes. Many authors thus believe that pair programming is much more efficient than individual programming [19,33]. All these benefits are because of the shoulder to shoulder technique followed in pair programming.

Despite its growing popularity, pair programming is rarely used for teaching an entire course due to the lack of reliable experimental data [28]. Also, there is no agreement on what factors should be considered while making pairs. Many of the studies pair students randomly [33] while some of the studies consider actual skills [5,18,19], perceived skills [17], and work ethics [41] of students for pairing. However, personality clash and disagreements are two real problems in pair programming [1], hence it is necessary to move a step ahead of skill-based pairing. Personality variables govern the interaction among programmers and influence the work style of every programmer significantly [38]. Since personality traits largely determine the compatibility of individuals, a pair with all the personality traits present has greater chances of attaining success [10]. Considering this, some authors have paired students on the basis of their personalities [7,37], but there were compatibility issues.

In this paper, we study the usefulness of pair programming as a method to teach an introductory programming course. The contributions of the study are the following:

- Keeping in mind the role of personality traits in determining the success of pair programming, we use a blend of skills and personality-based pairing mechanism. Apart from other traits of personality, we take students' attitudes toward programming into consideration while making pairs.
- A five-factor model comprising of skill levels of students, personality type, and attitude toward programming is considered for pairing students. To minimize conflicts and automate the pair formation, we propose a novel genetic algorithm-based approach.
- The experiment was conducted for one semester covering the entire course syllabus to analyze the effects of pair programming better.

At the end of the semester, the performance of students learning programming traditionally was compared with that of the students programming in pairs. In the end, we assessed the effectiveness of the genetic algorithm-based five-factor model by analyzing the compatibility of pairs with the help of a survey.

The rest of the paper is organized as follows. Section 2 explores the existing studies on pair programming. While exploring the existing studies, the methods adopted for making pairs are also listed. Section 3 explains the pair formation using a genetic algorithm. Here, we also explain the mechanism adopted to assign programming exercises dynamically on the basis of continuous assessment. Section 4 elaborates on the experimental details, materials, and methods used. In Section 5, we elaborate on the results of evaluations. Finally, we conclude and discuss future possibilities.

## 2 | RELATED WORK

### 2.1 | Pair programming in academics

Pair programming is used in teaching introductory programming courses since mid 1990's. In 1995, Larry Constantine experimented with teaching programming. It was observed that pair programming produced code faster, was efficient with respect to time, and resulted in students making few errors [9]. Similar outcomes were recorded by Laurie Williams in an Active Server Pages web programming class, but the students had prior experience with programming. The class size was also limited to 20 students only. A lot of other such studies were conducted in the next decade, focusing on various aspects of pair programming. For instance, in 2003, the impact of pair programming was assessed in a study that applied the technique on four sections of a university studying introductory programming course. Although the practice improved completion rate, pass rate, confidence, and enhanced quality of programs, nothing like compatibility was in the picture during those days [24]. Pairs were made solely on the basis of student's preferences.

Moving a step ahead of a simplistic comparison between collaborative and traditional learning, Lewis [22] compared the benefits of different types of collaborations with pair programming. It is evident that pair programming is a beneficial collaborative approach, but here the students' programming alone completed the assignments faster. This may be because the students under consideration were elementary school children, and results could differ in undergraduate courses. Similarly, Owolabi et al. [31] in 2013 compared the outcomes of teaching programming using Visual-Basic.Net by three different approaches, that is, lecture method, pair programming in the lab, and solo programming. Out of the three

approaches, pair programming and solo programming lead to better scores. However, no significant difference was observed between the performances of solo and pair programmers. The observations were based on 4 weeks duration only.

Contrary to this, there are experiments that assessed the effectiveness of the approach after experimenting for an entire semester and even longer. In a study, Ayub et al. [2] observed the effect of pair programming on slow- and fast-paced students separately for an entire semester. Although the benefits observed were for both the types, slow-paced learners were benefitted more compared to the faster ones. A similar study analyzed the impact of implementing pair programming on the basis of logical thinking. High logical thinkers were paired with low logical thinkers (LLT). The results showed significant improvement in the logical thinking abilities of LLT, who programmed in pairs [30]. It is said that pair programming not only improves the learning of the present programming course but also influences future performances [39]. Analyzed the long-term effects of pair programming for a time span of 2 academic years. It involved 2,468 students enrolled for an introductory computer science course. The observations clearly indicated the role of pair programming on present course grades, as well as future project scores.

Some other authors also considered observing the effectiveness for more than a semester. Ian McChesney [23] observed the impact of pair programming consecutively for 3 years, making it one of the longest studies on pair programming. Positive outcomes of pair programming were clearly visible on students' confidence as well as performance. Similar to all other studies, here too, pairs are made randomly. Several other studies examined the effectiveness and analyzed the benefits of using pair programming as a pedagogical tool [11,26,28,35]. In some experiments, a significant impact was seen [39], while in others there was no difference at all [31]. This is probably because the success of pair programming depends on several factors [11], one being understanding what makes the pairs work. Since efficient pairs in pair programming increase productivity [26], to view the exact impact of pair programming on undergraduate students, it is important to consider the pairing mechanism.

## 2.2 | Different approaches for making pairs

It is observed that the results of pair programming may vary if the pairs are not compatible enough. In a study, Williams et al. [41] tested the compatibility of pairs on a large scale with 1,350 students at a State University. By taking factors like personality type, skill level, and programming experience under consideration, overall, 93% compatibility was achieved. Salleh et al. [35], in a similar experiment, reported a total of 14 factors influencing the compatibility of pairs. Out of the 14 factors, actual skills and personality types influenced the success of pairing the most [35]. Several other authors have considered finding out the impact of these factors on performance and other aspects of pair programming. Hannay et al. [15] also stated that the performance of any team could be predicted with the personality types present in the pair/team. With 196 software professionals from three different countries, the authors made 96 pairs for testing. For assessment of personality type, they used the big five personality test. However, it was observed that personalities on the basis of big five personality test did not influence the compatibility of pairs much [15]. Contrary to this, some formal experiments were conducted in [29] taking into account 594 undergraduate students and recorded that some personality traits did influence the compatibility of pairs. They also used the big five personality test to identify various personality traits [29].

In this paper, we try to make the pairs after taking into account some factors that have already been studied, as well as some others that have not yet been considered for pair formation. From various studies, it is clear that factors like skill level and personality type influence the compatibility of pairs the most. Apart from skills and personality types, the experience in programming also affects the way pairs function [17]. These factors have been included separately or in combination with different pair programming approaches. One crucial aspect that influences the learning of any programming language is the attitude of the student toward programming. Despite attitude creating a significant difference in the learning [20], there is a scarcity of studies taking into consideration the mindset of students.

Apart from considering a blend of factors, including attitude toward programming, we use a genetic algorithm to make pairs automatically. A genetic algorithm is used for pair formation in pair programming for the first time. However, in problems similar to pairing, like stable marriage problem, the algorithm gives good results. Empirical evidence states that a genetic algorithm is an efficient local search algorithm to deal with stable marriage problem [13]. Authors have also used the algorithm for grouping the project supervisors with students [34]. Hence, we use a genetic algorithm to give us pairs that have a certain blend of the five factors under consideration. Since there is an evident role of skills, experience, and personality types, these three are given as input to the genetic algorithm.

## 3 | GENETIC ALGORITHM-BASED APPROACH FOR TEACHING PROGRAMMING

The paper uses a genetic algorithm-based approach for pair formation. Once the pairs are made, programming exercises are assigned to students and pairs of students dynamically on the basis of their performance in the previous programming session.

### 3.1 | Genetic algorithm-based pairing mechanism

Genetic algorithm is a search technique based on Charles Darwin's theory of evolution [21]. As the fittest individuals survive in the progression of genetic algorithm, most compatible students are finally paired together. Gradually, with the help of a genetic algorithm, the system moves toward a state where all the students have been paired optimally. The aim here is to stabilize the genetic algorithm once every pair has a minimum skill level and required personality traits. For this purpose, we first collected some data from the students. The following data is given as input to the genetic algorithm.

1. Total number of students in each in a programming class
2. Student profile describing the following five factors:
   - Actual skills: some basic information about their background knowledge and skill levels are gathered before the beginning of laboratory sessions. Since we are dealing with 1st-year students, the marks they score in 10th and 12th standards are used to determine their skill level. These scores are divided by 100 to get a value between 0 and 1. This is done to simplify the calculation of fitness.
   - Aptitude score: since all the students come from different boards of studies, to normalize any variations, an aptitude test was carried out. The test was of medium difficulty level and tested reasoning, numerical ability, and quantitative aptitude of students. The final score was recorded on a scale of 0–1 by the system.
   - Attitude toward programming: since attitude makes all the difference in learning anything new, it also becomes an important factor while making pairs. So, the students were asked to rank their tilt toward programming on a scale of 1–5. Following are the five ranks for determining attitude toward programming:
     1. I don't like programming at all
     2. I am not much interested in programming
     3. I neither like nor dislike programming
     4. I like writing programs
     5. I am passionate about programming

     The system reduces the responses on the scale of 0–1 to simplify further calculations.
   - MBTI personality type: MBTI personality test consists of a questionnaire that classifies any human being in one of the 16 personality types. These are represented with four-letter codes made up using the alphabets, that is, I (Introversion), E (Extraversion), S (Sensing), N (Intuition), T (Thinking), F (Feeling), J (judging), and P (perceiving). The sixteen types of personalities are "ISTP", "ISTJ," "ISFP," "ISFJ," "INTP," "INTJ," "INFP," "ENFJ," "ESTP," "ESTJ," "ESFP," "ESFJ," "ENTP," "ENTJ," "ENFP," and "ENFJ". Based on a questionnaire, any human being can be identified as one of the 16 personalities having certain types of traits dominating over others. Each personality type has distinct tendencies and is unique in its own way. Since every personality type has different strengths and weaknesses, it can be used as an indicator in pairing [14]. The system calculates the personality type and stores the assigned personality for each student.

     It is seen that beginners prefer working with a personality type different from their own [17] and different personality types when paired together perform better [37]. Considering the success and preferences, the algorithm gives preference to pairing different personality types. According to MBTI personality types, introversion and extraversion are two different personality traits. Similarly, Sensing-Intuition, Thinking-Feeling, and Judging-Perceiving are contrary [14]. Hence, to pair different personality types together, a pair of a student with "sensing" as a personality trait with one who is to the "intuitive" side according to MBTI would have a higher fitness score than two students with sensing characteristics. Similarly, the algorithm weighs the combination of "introversion-extraversion," "Thinkers-Feelers," and Judging-Perceiving" high.
   - Experience: it is also important to note whether the student has prior experience in programming or not. Pairing two experienced programmers can be great if you consider productivity as a measure of a successful pairing. On the other hand, pairing novices together can make it a difficult learning experience for the pair. By asking a simple question —"whether you have studied programming as a subject in previous classes or not," the system categorizes the programming class in two groups. One is of experienced, and the other is of novices with no prior knowledge of programming. If the

student is an experienced programmer, it is represented by 1 in the chromosome initialized, else the value is 0.

Following are the different stages of the genetic algorithm we have used for finding out the suitable pairs:

Step 1: Initialization: an array C, containing information about all the five factors (1—skills; 2—aptitude, 3—attitude toward programming; 4—Experience in programming, and 5—MBTI personality type) in consideration is initialized. Value encoding is used to store the data collected from the students [42].

The system creates chromosomes corresponding to every student who is going to learn programming in pairs. These chromosomes, when paired randomly, make the initial population. Figure 1 depicts a five-factor chromosome.

Initial population = $\{C_1, C_2 \ldots\ldots C_n\}$, where n is the number of students in each class.

Step 2: Calculate fitness: fitness is calculated for a pair of chromosomes using the following rules:

11. Initialize fitness $F_S$ to 0, where $s = 1, 2, \ldots n$;
12. For every pair $_{Pi}$, where $I = 1$ to $n/2$ if $n$ is even and $n + 1/2$ if $n$ is odd.

Repeat the following:

- For all the values in $C_s$[0 to 3] calculate the arithmetic mean. Here, $C$ represents the chromosome and $s = 0$ to $n$;
- Increment fitness $F_s$ by 1 if the arithmetic mean is greater than the threshold (0.5 in this case). This is done to ensure that students with similar skill levels have a higher probability of getting paired together [17].

13. Check the values of $C_s$[4] in the two chromosomes and increment the overall fitness by 1 if the two personality types are different since it is believed that different personality types lead to higher productivity when paired together [8,17,32]. The encoding scheme and scores assigned are listed in Table 1.
14. $F_s = F_s/6$;

2. Step 3: Selection: every student pair now has a fitness value assigned. Since, in most cases, it is not possible to get fit pairs at initialization, we move on to the next step. The entire population is divided into two sets. The fitset has good chromosomes represented by all the pairs whose fitness value is above 0.65. This is a compatibility threshold decided experimentally. The unfitset has those pairs that lie below the compatibility threshold. All the pairs in the unfitset are disintegrated and 20% of good chromosomes from any randomly selected pairs from fitset are also added to the pool.
3. Step 4: Crossover: new pairs are formed using the chromosomes in the unfitset and those that qualify the compatibility threshold are added to the fitset. Crossover is performed till the unfitset is empty or for up to 800 iterations, whichever is earlier. If the crossover is performed 800 times, the system separates all the pairs from both the sets and starts all over again. Figure 2 shows the flow chart of pair generation using a genetic algorithm.

**TABLE 1** Encoding scheme and corresponding scores

| Position in C | Encoding | Related score |
|---|---|---|
| C[0] | Actual skills | 0 if mean <0.5, else 1 |
| C[1] | Aptitude | 0 if mean <0.5, else 1 |
| C[2] | Attitude | 0 if mean <0.5, else 1 |
| C[3] | Experience | 0 if mean <0.5, else 1 |
| C[4] | MBTI personality type | 1 if the pair has different personality types, else 0 |

Mutation is the fifth step of a genetic algorithm. However, in this case, it is not much applicable because we are representing compatibility factors as chromosomes and it cannot be modified for a student. There are chances of not getting optimal pairs too, because it depends on the types of students we get every year. In such a case, since the problem is to find good pairs from the existing students only, we can lower down the compatibility threshold. Figure 3 depicts the modified genetic algorithm used.

## 3.2 | Recommending programming exercises dynamically

We developed a desktop application to assign programming exercises to students working individually and those learning in pairs. Various studies explore pair

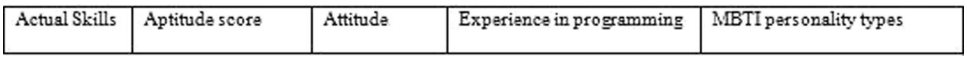


**FIGURE 1** 5 Factor chromosome

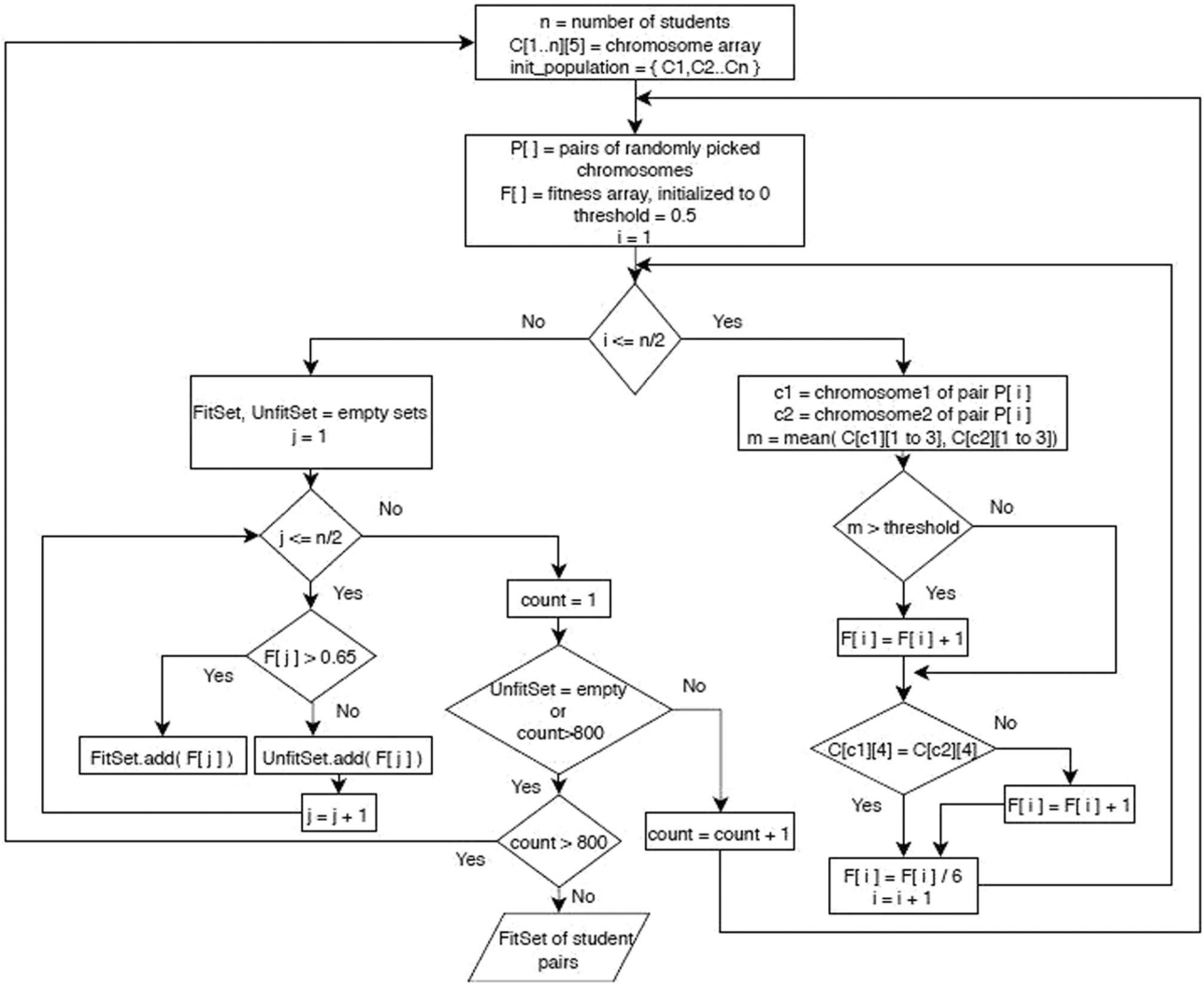


**FIGURE 2** Flow chart for pair generation using a genetic algorithm

programming and other approaches to teaching programming [16]. However, only a few choose to automate the process of assigning programming exercises dynamically. This was necessary to maintain the uniformity of difficulty level of programming exercises among all the students. For each concept to be taught, we listed a few programming exercises. In the first lab, all the students are given the same basic exercises. On the basis of the assessment done at the end of the lab session, every student/pair is given a score out of 10. This score was assigned on the basis of the readability, functionality, and quality of code. A system for assigning exercises is developed in which you get the exercises of the next programming concept only when you have a score of 7 or higher in the present concept. If the student/pair scores below 7, he/she is recommended some other programming exercise from the same concept. The system selects an exercise randomly using a genetic algorithm-based random number generator. The exercises are assigned randomly from a set of programming problems of the same difficulty level until the student scores at least 7. The list of programs is attached in Appendix I. The students who were suggested the programming problems of the same concept were given extra time in the laboratory to solve programming problems of the next concept. This way, all the students were assessed uniformly and performed similar programming problems.

## 4 | MATERIALS AND METHODS

During the Autumn 2019 semester, 1st-year undergraduate students of Netaji Subhas University of Technology are given an introductory programming course. We selected a total of 171 students enrolled in computer engineering department. The undergraduate students are divided randomly into sections to ensure access to laboratory and other resources for every student. Section 1

```
ALGORITHM
1: Step 1. Initialize: n = number of students
2: let C [1...n][ 5 ] be a new chromosomes array
3: Initial_population={C1, C2.....Cn}

4: Step 2. Fitness: Pairs, P [ ], formed with randomly picked chromosomes from
initial population,
   P [1...n/2] = Random_Pairs (initial_population)
5: let F [1....n/2] be a new Fitness array, initialized to 0
6: let threshold = 0.5
   7: for i = 1 to n/2 do
        8: c1 = chromosome1 of P[i]
        9: c2 = chromosome2 of P[i]
        10: m = arithmetic_mean ( C [ c1 ][ 1 to 3 ], C [ c2 ][ 1 to 3 ] )
          11: if m > threshold then
                12: F [i] = F [ i ] + 1
        13: end if
   14: end for
   15: if C [c1][ 4 ] ≠ C [ c2 ][ 4 ] then
         16: F [i] = F [i] + 1
   17: end if
18: F [i] = F [ i ] / 6

19: Step 3. Selection: FitSet, UnfitSet
    20: for i = 1 to n/2 do
          21: if F [i] > 0.65 then
                22: FitSet.add (P [ i ])
          23: end if
          24: else
              25: UnfitSet.add (P [ i ])
          26: end else
    27: end for

28: Step 4. Crossover: let count = 1
        29: while UnfitSet ≠ empty and count ≤ 800
        30: repeat Step 2 and Step 3
              31: count = count + 1
        32: end while
        33: if count ≥ 800 then
              34: go to Step 1
        35: end if

36: Recommend pairs
```

**FIGURE 3** Modified genetic algorithm

comprising of 83 students was arbitrarily selected as the control group and Section 2 with strength of 88 students was chosen as the experimental group. Since the students are selected by the university through the scores of a common entrance exam, those securing almost similar scores are offered admissions to computer science courses in the university. However, just to be sure, we compared the average scores of Secondary school exam (10th grade) and Senior secondary school exams (12th grade) of students from both the sections. The average scores given in Table 2 are quite similar and hence we can say skills and intelligence quotient are equally distributed among all the students of experimental and control group.

Control group programmed in a traditional manner with each student programming alone. There was no sharing of resources or conversations allowed. On the other hand, the students of the experimental group were made to program in pairs. Before the beginning of the lab

**TABLE 2** Average scores of students from two sections

| Section | Average score out of 100 (10th) | Average score out of 100 (12th) |
|---|---|---|
| Section 1 | 84.42 | 81.08 |
| Section 2 | 83.70 | 82.41 |

sessions, every student from the experimental group was asked to create a profile on a desktop application. Here, they were asked to answer a few questions about their past study record. In addition to the basic information, they also underwent two tests: (a) aptitude test and (b) MBTI test. The aptitude test comprised of 20 questions to test the arithmetic ability, logical reasoning, and data interpretation skills. MBTI test consisted of some easy to answer questions that are required for analyzing the personality traits of any human being.

On the data from the experimental group, we applied a genetic algorithm to generate suitable pairs. These students were asked to solve their programming exercises with the assigned partner for the entire semester. For 14 weeks, all the students received different programming exercises every week. Students from the control group were given programming exercises to be completed in 120 min lab session. However, the experimental group was supposed to complete the exercise in 90 min. At the end of every lab session, the solutions submitted by every student/pair were evaluated for functionality and readability. The programming exercises were identical in terms of difficulty, and so was the final exam for all the students. The final practical examination assessed students' knowledge of programming concepts and expertise in coding.

At the end of the course, the scores of programming exercises and that of the final exam served as dependent measures. Apart from these two, the students from the experimental group were also asked to submit a short peer evaluation to judge the compatibility of pairs. Figure 4 represents the distribution of compatibility scores given by students to their partners.

The final scores of the experimental group and control group were then compared. The process of research followed is depicted in Figure 5.

## 5 | RESULTS

To evaluate the effect of pair programming as a tool for an undergraduate programming course, we compared the final scores of the experimental and control group. At the end of the programming course, students from the experimental group that performed all the programming tasks in pairs and the control group comprising of individual programmers appeared in the final assessment. The final exam tested the conceptual understanding and ability to program independently. Every student was given a score out of 100. Using one-way analysis of variance (ANOVA), with a $p = .831614$ and $f$ value of 0.04535, the difference between the final scores of the experimental

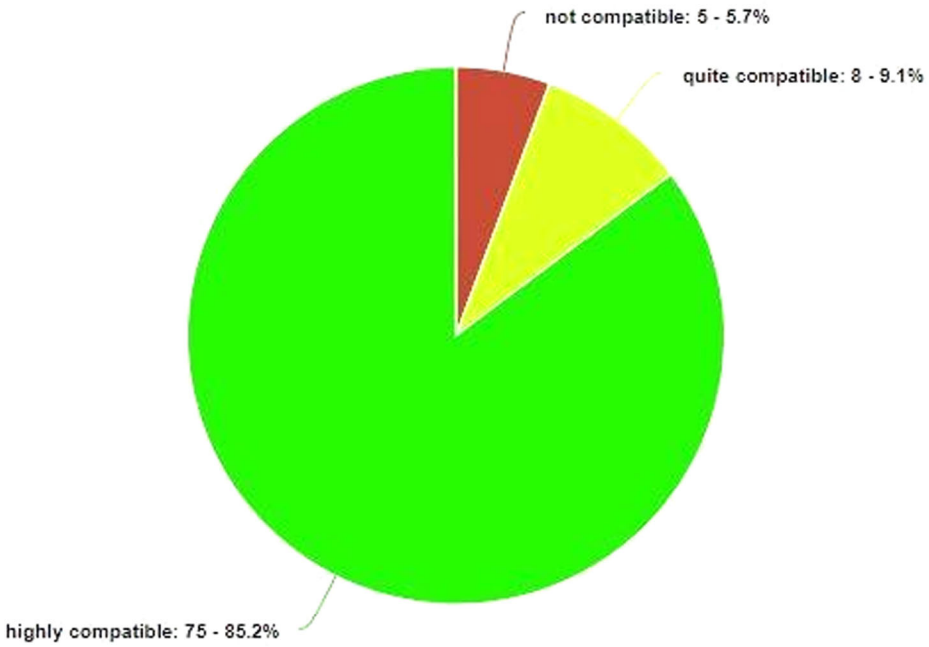


**FIGURE 4** Responses for compatibility scores

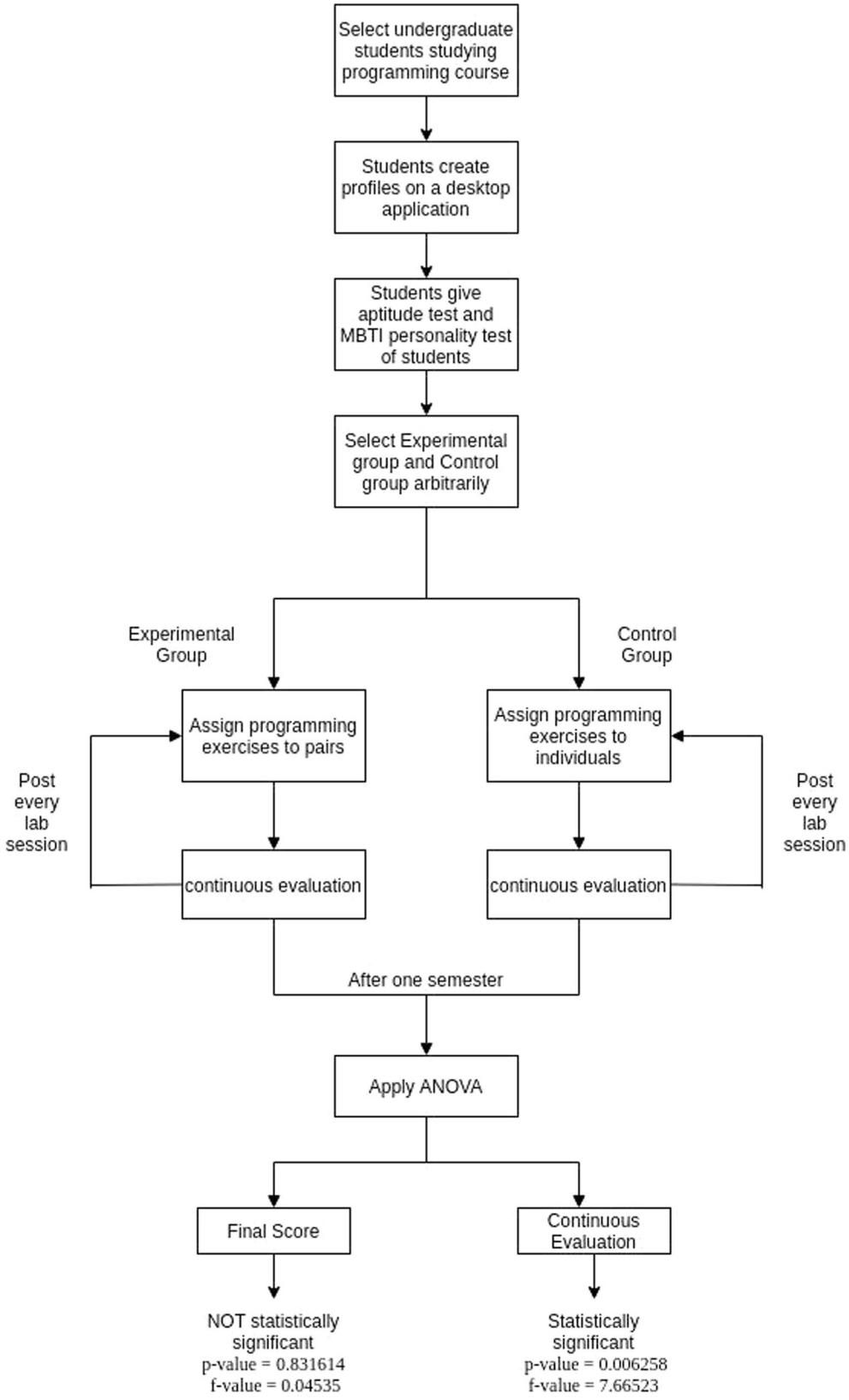


**FIGURE 5** Flow chart depicting the procedure of research

**TABLE 3** Distribution of responses

| Compatibility score | Participation (Yes) | Participation (No) | Contribution (Yes) | Contribution (No) | Cooperation (Yes) | Cooperation (No) |
|---|---|---|---|---|---|---|
| 0 | 1 | 4 | 3 | 2 | 0 | 5 |
| 1 | 5 | 3 | 8 | 0 | 6 | 2 |
| 2 | 74 | 0 | 74 | 0 | 74 | 0 |

and control group was not found to be statistically significant. However, to assess the impact of pairing on every lab sessions' performance, we also applied ANOVA on the average scores of programming exercises too. The results showed statistically significant differences in the performance of the experimental and control group at $f = 7.66523$ and a $p$ value of .006258.

Apart from observing the impact of pair programming, we also assessed the compatibility of pairs. It was possible that the pairs we made on the basis of five factors did not work in a balanced manner. In this case, pair programming would not meet its purpose. This was checked with short peer evaluation and compatibility survey. All the 88 students from the experimental group participated in the survey. They were asked a few simple questions to assess participation, contribution, and cooperation. They were also requested to rate their compatibility with their partners on a 3-point Likert scale. Almost 85% of the students found that their partners were highly compatible. Overall, it can be seen that the five-factor-based genetic algorithm model leads to 94.31% compatibility, higher than [41] using a different approach. This clearly indicates that pairing using a genetic algorithm gave quite good results.

To understand the dependencies among variables, we applied the $\chi^2$ test on the data distribution [25]. It was done at a level of significance 0.95, indicating the critical value of $\chi^2$ distribution to be 18.307 at 10 degrees of freedom. The division on the basis of different categories is given in Table 3.

The test statistic value of the $\chi^2$ test being 151.142 shows a correlation between different parameters. This shows that the compatibility of partners, participation, cooperation, and contribution are interdependent factors for pairs in this experiment. Apart from the psychometric questions, all the students from the experimental as well as control group were asked to describe their experiences using a few words. While most of the students from the control group described the lab sessions with phrases like "Normal," "Not interesting," "As expected," and so forth, the phrases emerging from experimental groups suggested something different. Repeated occurrences of words like "Amazing," "Enjoyable," "Easy," and "Good experience" indicated that the experimental group had better classroom experience. Also, according to the observations made by the instructor, the students from the experimental group completed their exercises faster when compared to the students from the control group. Taking into consideration all these outcomes, the results of the study encourage the use of pair programming as a pedagogical tool.

## 6 | CONCLUSION

We experimented with pair programming for an introductory course of programming offered to 1st-year computer engineering students. Eighty-three students in the control group programmed individually. On the other hand, a total of 88 students in the experimental group were made to program in pairs for the entire duration of the course. Both the groups received programming exercises dynamically and the level of difficulty was the same. The evaluation was also done with programming challenges of the same difficulty level for both the groups. Analysis of student feedback and experiences of the instructor with the experimental group have been favorable. With a significant difference in scores of programming exercises, it was observed that the quality of code was much better in the case of pair programming. The instructor observed that the students communicated with each other more and appeared to learn and code faster. With approximately 94% of the students finding their pairing compatible, using a genetic algorithm with all the five factors into consideration seems to be a good idea. The study indicates that pair programming has its own set of benefits when used as a pedagogical tool for teaching introductory programming courses. Teachers trying to pair students on the basis of some factors can follow the genetic algorithm model to automate the pair generation. Apart from teachers, the students can utilize the system we develop for assigning programming exercises dynamically for mastering a specific concept in programming.

The study focuses on assessing the overall impact of pair programming and evaluating the compatibility

between pairs generated using a genetic algorithm. However, programming in pairs also influences factors like logical thinking differently in different students. Different types of learners, like slow-paced and fast learners, may receive the benefits of pair programming differently. Thus, the influence of pair programming may also differ from student to student. In the present study, this aspect is not taken into consideration. In addition to this, although the study was conducted for the entire course duration, it may not be useful for assessing the long-term effects of pair programming.

In the future, the authors would like to study how the benefits of pair programming can vary from student to student. We would also try to evaluate the long-term effects pair programming leaves when used as a pedagogical tool in the introductory programming course. Since pair programming is gaining popularity, we also plan to develop a website that integrates the entire process of pair formation using the genetic algorithm model proposed.

## DATA AVAILABILITY STATEMENT

Data in the summarized form is available in the article tables and figures. The entire data that support the findings of this study are available from the corresponding author upon reasonable request.

## ORCID

*Kanika Tehlan* http://orcid.org/0000-0002-8352-9940
*Pinaki Chakraborty* http://orcid.org/0000-0002-2010-8022

## REFERENCES

1. B. Andrew and N. Nagappan, *Pair programming: What's in it for me?*, Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, 2008, pp. 120–128.
2. M. Ayub et al., *Utilising pair programming to enhance the performance of slow-paced students on introductory programming*, J. Technol. Sci. Educ. **9** (2019), 357–367.
3. K. Beck, *Extreme programming explained: Embrace change*, 2nd ed., Addison-Wesley, Reading, MA, 2005.
4. T. Bipp, A. Lepper and D. Schmedding, *Pair programming in software development teams—An empirical study of its benefits*, Inform. Software Tech. **50** (2008), 231–240.
5. G. Canfora et al., *Confirming the influence of educational background in pair-design knowledge through experiments*, Proceedings of ACM Symposium Applied Computing, 2005, pp. 1478–1484.
6. G. Canfora, A. Cimitile and C. A. Visaggio, *Working in pairs as a means for design knowledge building: An empirical study*, Proceedings of 12th IEEE International Workshop on Program Comprehension, 2004, pp. 62–68.
7. K. S. Choi, *A discovery and analysis of influencing factors of pair programming*, PhD dissertation, Department of Information Systems, New Jersey Institute of Technology, 2004.
8. K. S. Choi, F. P. Deek and I. Im, *Exploring the underlying aspects of pair programming: The impact of personality*, Inform. Software Tech. **50** (2008), 1114–1126.
9. L. L. Constantine, *Constantine on peopleware*, Yourdon Press, Englewood Cliffs, NJ, 1995.
10. A. J. Dick and B. Zarnett, *Paired programming & personality traits*, Proceedings of third International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2002), pp. 82–85.
11. T. Dyba et al., *Are two heads better than one? On the effectiveness of pair programming*, IEEE Software **24** (2007), 12–15.
12. Z. Franz and L. Prechelt, *Does pair programming pay off?* Rethinking productivity in software engineering, Apress, Berkeley, CA, 2019.
13. B. Genc et al. *Finding robust solutions to stable marriage*, Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, 2017, pp. 631–637.
14. V. P. Goby, *Personality and online/offline choices: MBTI profiles and favored communication modes in a Singapore study*, Cyberpsychol. Behav. **9** (2006), 5–13.
15. J. E. Hannay et al., *Effects of personality on pair programming*, IEEE T Software Eng. **36** (2009), 61–80.
16. S. Kanika, Chakraverty and P. Chakraborty, *Tools and techniques for teaching computer programming: A review*, J. Educ. Technol. Syst. **49** (2020), 170–198.
17. N. Katira et al., *On understanding compatibility of student pair programmers*, Proceedings of thirty fifth SIGCSE Technical Symposium on Computer Science Education, 2004, pp. 7–11.
18. N. Katira, L. Williams and J. Osborne, *Towards increasing the compatibility of student pair programmers*, Proceedings of twenty seventh International Conference on Software Engineering, 2005, pp. 625–626.
19. C. Lan and B. Ramesh, *An exploratory study on the effects of pair programming*, Proceedings of eighth International Conference on Empirical Assessment in Software Engineering, 2004, pp. 21–28.
20. K. M. Y. Law, C. S. Victor and Y. T. Yu, *Learning motivation in e-learning facilitated computer programming courses*, Comput. Educ. **55** (2010), 218–228.
21. D. Lawrence, *Handbook of genetic algorithms*, Van Nostrand Reinhold, New York, 1991.
22. C. M. Lewis, *Is pair programming more effective than other forms of collaboration for young students?* Comput. Sci. Educ. **21** (2011), 105–134.
23. I. McChesney, *Three years of student pair programming: Action research insights and outcomes*, Proceedings of the forty seventh ACM Technical Symposium on Computing Science Education, 2016, pp. 84–89.
24. C. McDowell et al., *The impact of pair programming on student performance, perception and persistence*, Proceedings of the Twenty-fifth International Conference on Software Engineering, 2003, pp. 602–607.
25. M. L. McHugh, *The chi-square test of independence*, Biochem. Med. **23** (2013), 143–149.
26. N. Mohanraj, S. Akshaya and A. Palanikumar, *Pair recommendation for pair programming*, Int. J. Emerg. Res. Dev. **9** (2019), 55–58.
27. M. M. Mueller and O. Hagner, *Experiment about test-first programming*, IEE Proc.—Softw. **149** (2002), 131–136.

28. J. Nawrocki and A. Wojciechowski, *Experimental evaluation of pair programming*, Proceedings of European Software Control and Metrics Conference, 2001, pp. 269–276.
29. S. Norsaremah, E. Mendes and J. Grundy, *Investigating the effects of personality traits on pair programming in a higher education setting through a family of experiments*, Empir. Softw. Eng. **19** (2014), 714–752.
30. M. Othman et al., *The impact of pair programming on students logical thinking: A case study on higher academic institution*, SMRJ **16** (2019), 85–98.
31. J. Owolabi et al., *Effects of solo and pair programming instructional strategies on students' academic achievement in visual-basic.net computer programming language*, Global Sci. Technol. Forum J. Comput. **3** (2013), 108–111.
32. A. Raza, L. F. Capretz and Z. Ul-Mustafa, *Personality profiles of software engineers and their software quality preferences*, Int. J. Inform. Sys. Soc. Chnage **5** (2014), 77–86.
33. F. J. Rodríguez, K. M. Price and K. E. Boyer, *Exploring the pair programming process: Characteristics of effective collaboration*, Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, 2017, pp. 507–512.
34. H. O. Salami and E. Y. Mamman, *A genetic algorithm for allocating project supervisors to students*, Int. J. Intell. Syst. Appl. **8** (2016), 51–59.
35. N. Salleh, E. Mendes and J. Grundy, *Empirical studies of pair programming for CS/SE teaching in higher education: A systematic literature review*, IEEE T Software Eng. **37** (2011), 509–525.
36. G. Salomon, *Distributed cognitions: Psychological and educational considerations*, Cambridge Press, Cambridge, MA, 1993.
37. P. Sfetsos et al., *Investigating the impact of personality types on communication and collaboration-viability in pair programming —An empirical study*, Proceedings of seventh International Conference on Extreme Programming and Agile Processes in Software Engineering, 2006, pp. 43–52.
38. B. Shneiderman, *Human factors in computer and information systems*, Software Psychology, Winthrop Publishers, Cambridge, MA, 1980.
39. M. O. Smith, G. Andrew and D. Andrew, *Long term effects of pair programming*, IEEE T Educ. **61** (2017), 187–194.
40. L. Williams et al., *Strengthening the case for pair programming*, IEEE Software **17** (2000), 19–25.
41. L. Williams et al., *Examining the compatibility of student pair programmers*, Proceedings of AGILE—2006 Conference, 2006, pp. 1–10.
42. A. H. Wright, *Genetic algorithms for real parameter optimization*, Foundations of genetic algorithms, 1st ed., Elsevier, San Mateo, CA, 1991.


## AUTHOR BIOGRAPHIES

**Kanika Tehlan** is a Senior Research Fellow at Netaji Subhas Institute of Technology (NSIT). She is working on developing educational tools, e-learning models, and improving the quality of learning using different environments.

**Shampa Chakraverty** is a professor at Netaji Subhas University of Technology. Her research areas include analysis of sentiment, emotion, and human language and e-learning, information retrieval, and engineering pedagogy.

**Pinaki Chakraborty** received his B. Tech. from Indraprastha University and his M. Tech. and Ph. D. from Jawaharlal Nehru University. He is an assistant professor at Netaji Subhas University of Technology. His area of research includes systems software and educational software.

**Shradha Khapra** is a B. Tech. student studying Computer Engineering at Netaji Subhas Institute of Technology. She is actively exploring the different areas of educational software, collaborative learning, and pedagogy.

**How to cite this article:** Tehlan K, Chakraverty S, Chakraborty P, Khapra S. A genetic algorithm-based approach for making pairs and assigning exercises in a programming course. *Comput Appl Eng Educ.* 2020;1–14. https://doi.org/10.1002/cae.22349

## APPENDIX

A. **INITIAL LEVEL (CONDITIONAL STATEMENTS)**

1. Write a program to add two integers.
2. Write a program to multiply two floating point numbers.
3. Write a C program to check whether a given number is positive or negative.
4. Write a C program to accept two integers and check whether they are equal or not.
5. Write a program to find the largest among three numbers.
6. Write a program to determine if a number is odd or even.
7. Write a program to determine if a number is prime or not.
8. Write a C program to find whether a given year is a leap year or not.
9. Write a C program to read the age of a candidate and determine whether it is eligible for casting his/her own vote.

10. Write a C program to check whether a triangle is Equilateral, Isosceles, or Scalene.

A. **INTEGER ARITHMETIC**

11. Write a program to find LCM (lowest common multiple) of two numbers.
12. Write a program to find GCD (greatest common divisor) of two numbers.
13. Write a program to print the first ten fibonacci numbers.
14. Write a program to print all prime numbers between 1 to 500.
15. Write a program to swap two numbers without using the third number.

A. **LOOPS—while loop**

16. Write a C program to print all natural numbers from 1 to n. - using while loop.
17. Write a C program to print all natural numbers in reverse (from n to 1). - using while loop.
18. Write a C program to print all alphabets from a to z. - using while loop.
19. Write a C program to print all even numbers between 1 to 100. - using while loop.
20. Write a C program to print all odd number between 1 to 100.- using while loop.

**For loop**

21. Write a program in C to display the first 10 natural numbers (using for loop).
22. Write a C program to calculate the factorial of a given number (using for loop).
23. Write a C program to find the sum of first 10 natural numbers (using for loop).
24. Write a program in C to display the cube of the number upto given an integer (using for loop).
25. Write a C program to determine whether a given number is prime or not (using for loop).

A. **ARRAYS**

26. Write a program in C to store elements in an array and print it.
27. Write a program to search for a number in an array.
28. Write a program to merge two arrays.
29. Write a program in C to read n number of values in an array and display it in reverse order.
30. Write a program in C to find the sum of all elements of the array.
31. Write a program in C to copy the elements of one array into another array.
32. Write a program in C to print all unique elements in an array.
33. Write a program in C to find the maximum and minimum element in an array.
34. Write a program in C to delete an element at desired position from an array.
35. Write a program in C to find the second smallest element in an array.

A. **MATRICES**

36. Write a program in C for multiplication of two square Matrices.
37. Write a program in C for addition of two Matrices of same size.
38. Write a program in C for subtraction of two Matrices.
39. Write a program in C to find sum of rows and columns of a Matrix.
40. Write a program in C to accept two matrices and check whether they are equal.

A. **FUNCTION**

41. Write a function to swap two numbers
42. Write a program in C to check a given number is even or odd using the function.
43. Write a program in C to convert decimal number to binary number using the function.
44. Write a program in C to get the largest element of an array using the function.
45. Write a program in C to find the square of any number using the function.

A. **SEARCHING AND SORTING**

46. Write a program to sort the elements of an array in ascending order.
47. Write a program to demonstrate binary search.
48. Write a program to search for a number in an array using binary search.
49. Write a program to sort a list using bubble sort algorithm.
50. Write a program to sort a list of elements using selection sort algorithm.

A. **RECURSION**

51. Write a program in C to print first 50 natural numbers using recursion.
52. Write a program in C to calculate the sum of numbers from 1 to n using recursion.
53. Write a program in C for binary search using recursion.
54. Write a program in C to find the first capital letter in a string using recursion.
55. Write a program in C to copy one string to another using recursion.

I. **INHERITANCE**

56. Write a program to demonstrate the use of inheritance.
57. Write a program to demonstrate the use of multiple inheritance.
58. Write a program to demonstrate the use of multilevel inheritance.
59. Write a program to demonstrate hierarchical inheritance.

60. Write a program to demonstrate private simple inheritance.

A. **STRUCTURE**

61. Write a program to store and print the roll no., name, age, and marks of a student using structures.
62. Write a program to add two distances in inch-feet using structure. The values of the distances have to be taken from the user.
63. Write a program to add, subtract, and multiply two complex numbers using structures to function.

    Write a structure to store the name, account number, and balance of customers (more than 10) and store their information.
64. Write a function to print the names of all the customers having balance less than $200.
65. Write a function to add $100 in the balance of all the customers having more than $1000 in their balance and then print the incremented value of their balance.

A. **POINTER**

66. Write a program in C to show the basic declaration of pointer.
67. Write a program in C to demonstrate the use of & (address of) and * (value at address) operator.
68. Write a program in C to add two numbers using pointers.
69. Write a program in C to find the maximum number between two numbers using a pointer.
70. Write a program in C to store n elements in an array and print the elements using pointer.

A. **FILE HANDLING**

71. Write a program in C to create and store information in a text file.
72. Write a program in C to read an existing file.
73. Write a program in C to write multiple lines in a text file.
74. Write a program in C to Find the Number of Lines in a Text File.
75. Write a program in C to count a number of words and characters in a file.

A. **MISCELLANEOUS**

76. Write a program to demonstrate the use of default arguments.
77. Write a program to demonstrate function overloading.
78. Write a program to demonstrate the use of objects.
79. Write a program to demonstrate the use of new and delete operator.
80. Write a program to demonstrate the use of operator overloading.
81. Write a program to demonstrate the use of virtual functions.
82. Write a program to handle the division by zero exception.
83. Write a program to demonstrate the use of enumerated data type.
84. Write a program that uses command line argument.
85. Write a program to read and write a binary file.