# Improving Computer Programming Competence: The Parikshak Approach

**Dr. M. Sasikumar**
CDAC Mumbai
sasi@cdac.in

## Programming competence: challenges

There are a number of concerns about the quality of students coming out of our engineering and MCA colleges, in the area of information technology or computer science. Lack of programming proficiency is one of the major ones. Ability to design and implement a computer program is a key competence required of an IT professional. He/she should also be able to debug a given program and get it working as per the specifications.

This is important for many reasons. The basis of the IT and CS fields is programming a computer to do a given task. Even if one is looking at higher level tasks like design, architecture, etc., a strong foundation in programming is a must to do these well. Tasks like testing and debugging is also very hard without this base.

Usually when programming languages are taught, the focus is on the language features, and covering as many intricate constructs of the language as possible. So, a C programming student is expected to understand what **(a+*b) and the like means, no matter if any sensible programmer is likely to write such code. Students can give some kind of responses if asked what a particular construct does, but are lost in recognising the right use of such constructs in a program, or in writing a program using that construct. In our interviews, I used to ask students short programming problems like counting number of 0s in an array, finding length of a string, etc. I had named them 20-20 problems taking a cue from the game of cricket. The core of such programs is just 2-4 lines of code. Very few students are in a position to even attempt such programs, and fewer still can produce a sensible code.

Programming needs to be taught as problem solving in courses. Programming constructs, libraries and the like are to be introduced as tools for solving problems. This is rarely done, with the stress on the features and intricacies of the language, with the result that students know the language constructs, but find it difficult to use them correctly in their programs. The places where students learn to code are the programming labs.

## Programming labs in colleges

Most college programs, say BE/ME/MCA include programming labs. Sometimes these are co-located with the corresponding theory course, and sometimes, one is left to learn a language only using a lab course. These lab courses are meant to achieve the capability of writing programs for given problems using a specified programming language. However, this objective is rarely achieved for multiple reasons. The labs are usually of a fixed duration with everyone crowding around. The teacher will get very limited time to look at each student's work and assess it systematically. Beyond the lab time, the work is accessible neither for the faculty nor the student. The challenges of mass copying are, well, the less said the better.

Assessing a program provided by a student is not a trivial task. Ideally the teacher has to:

- compile the program and check for basic errors,
- run the program with adequate number of test cases to see if it is working as expected,
- check the quality of the code (layout, use of variables, etc), and documentation,
- and preferably check the efficiency of the algorithm used in terms of time and space.

Over and above this, if the problem involves UI design or external interfaces, then additional checks for these are also required to be done.

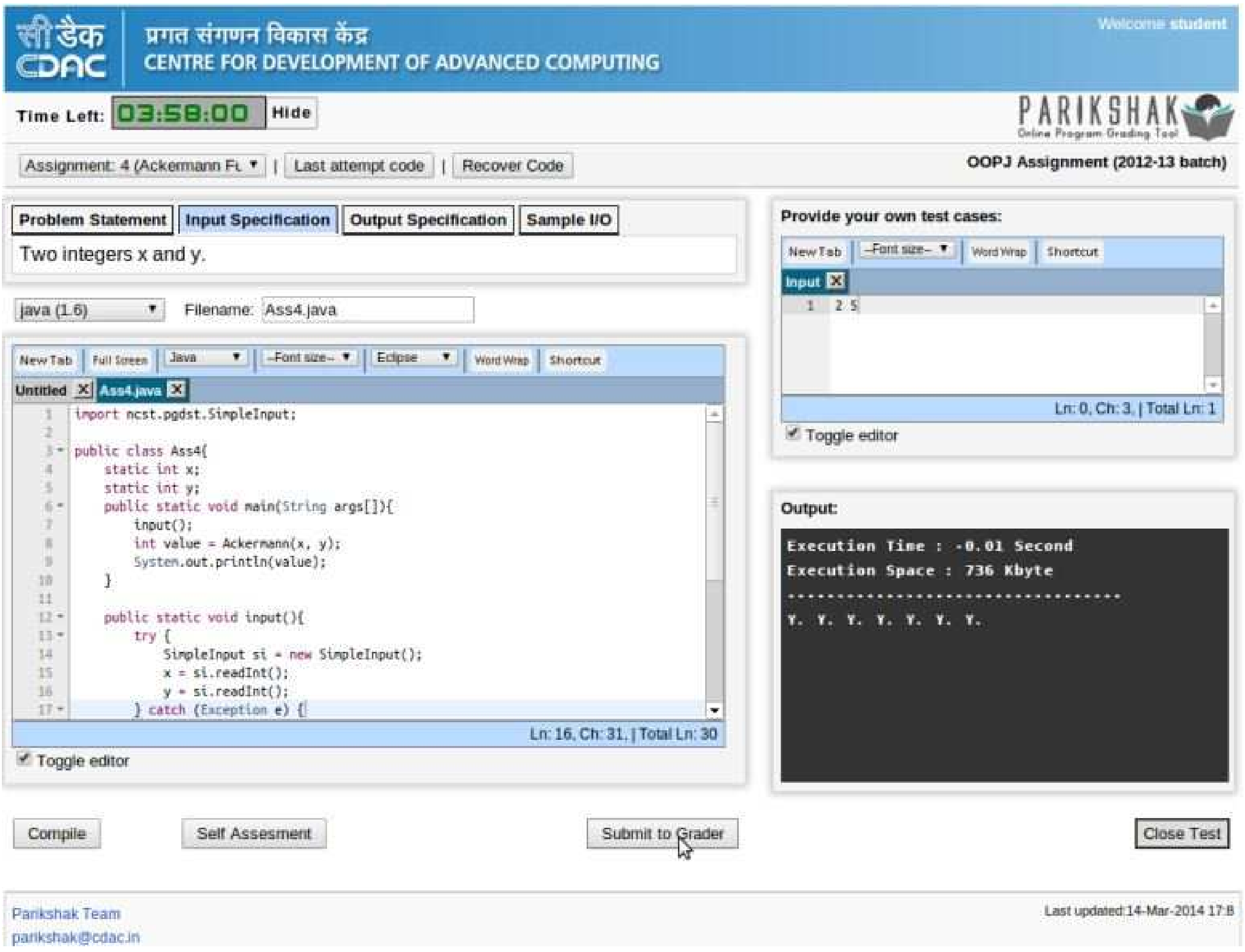
All these are too difficult to carry out effectively within the limited lab hours. Without regular and continuous assessment and feedback, the labs are of limited use in teaching/learning effective programming practices.

## What is Parikshak?

It is in this context, we would like to introduce a solution (actually, more of an approach) called Parikshak to help in this. Parikshak is a system designed and implemented and actively used at NCST, the predecessor of CDAC Mumbai, from late 1980s!! NCST faculty recognised the need for programming competence from then, and ensured that every student coming out of their training programs possessed this competence.

Parikshak, the software system which supported this, is essentially a program grading system. It has a simple editor which the user can use to write programs, and when ready, it can be submitted to grading. The grader evaluates the program against a number of pre-set test cases, and reports how many cases the program ran correctly, and how many failed. The user can revise the program and resubmit, repeating this cycle any number of times, till the time allocated for that problem is over, or he is satisfied whichever is earlier. See figure 1.

A registered teacher can create questions and exams, enrol students, and monitor the test while in conduct as well as after the exam. See figure 2. Students version of code is maintained every time a submit is made, enabling the teacher to track the student's development of the solution. The system offers logins for students to check their status, and to take any exam assigned to them.



Time Left: 03:58:00

Assignment: 4 (Ackermann Fu

OOPJ Assignment (2012-13 batch)

Two integers x and y.

java (1.6)    Filename: Ass4.java

```
import ncst.pgdst.SimpleInput;

public class Ass4{
    static int x;
    static int y;
    public static void main(String args[]){
        input();
        int value = Ackermann(x, y);
        System.out.println(value);
    }

    public static void input(){
        try {
            SimpleInput si = new SimpleInput();
            x = si.readInt();
            y = si.readInt();
        } catch (Exception e) {
```

Ln: 16, Ch: 31, | Total Ln: 30

Provide your own test cases:

```
2 5
```

Ln: 0, Ch: 3, | Total Ln: 1

Output:

```
Execution Time : -0.01 Second
Execution Space : 736 Kbyte
-----------------------------------
Y. Y. Y. Y. Y. Y. Y.
```

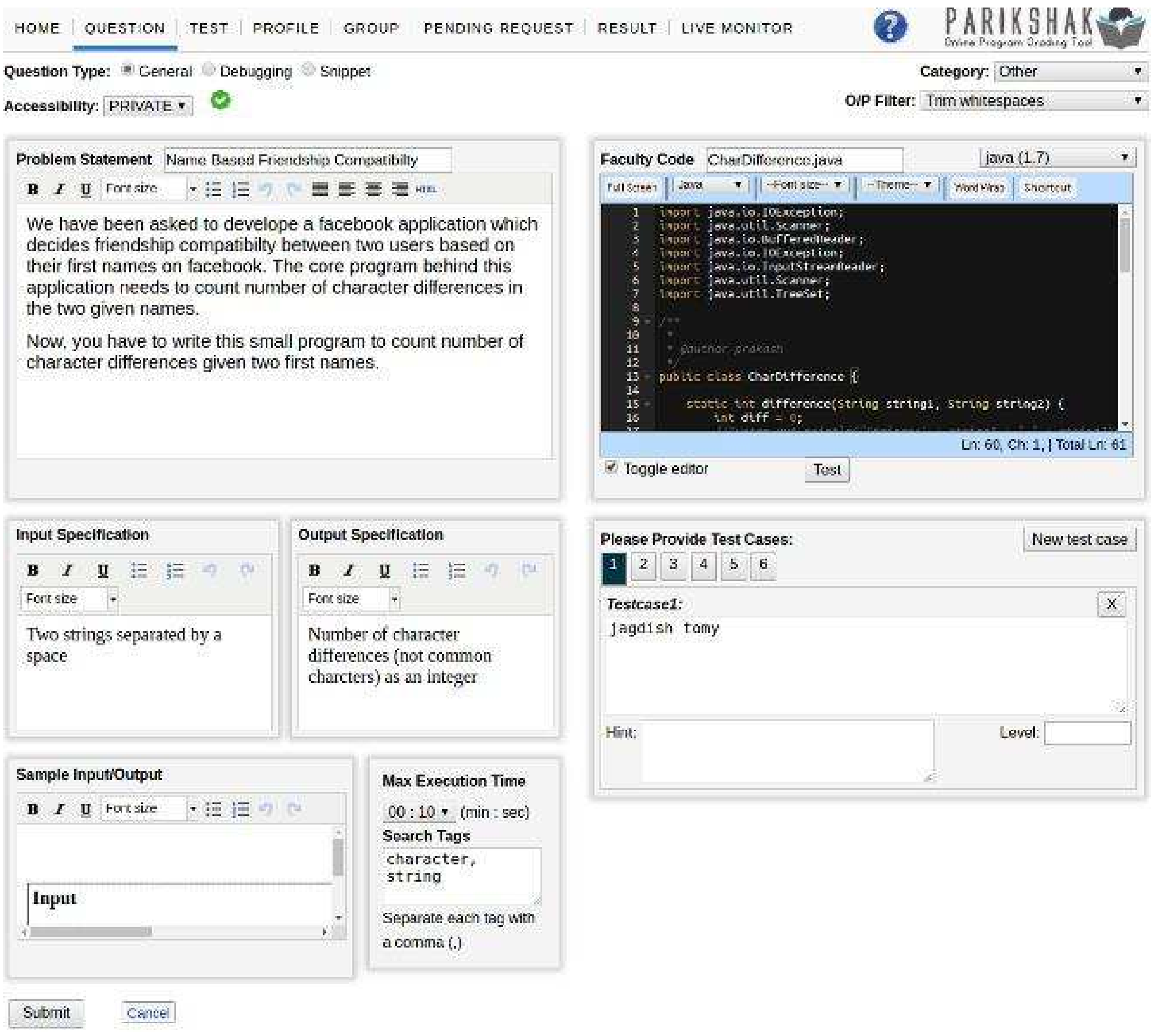
Parikshak Team
parikshak@cdac.in

Last updated:14-Mar-2014 17:8

*Figure 1: Student Interface*

The system which commenced as a set of shell scripts in the late 80s at NCST – remember there was no GUI or Web then -- has evolved over the years into a comprehensive web based system with rich features today, as the figures show. Recommended mode of usage is to access it on the web from the URL parikshak.in (requires a license), from any computer or tablet running any standard web browser. No machine specific installation is required. Where Internet access is a constraint, an option to setup a local installation over campus LAN is also available.

The web based version has been in operation only for a few years, mostly in silent mode though, with hardly any publicity. As of now, the portal has over 8000 registered users from over 70 organisations including a few companies and many academic institutions. About 100 questions are available open to all. Taking note of the suggestions from many of our users, and friends who have gone through Parikshak based examinations with us, we are now opening up the system to interested organisations.



Question Type: [✓ General] Debugging Snippet

Accessibility: PRIVATE

Category: Other

O/P Filter: Trim whitespaces

Problem Statement: Name Based Friendship Compatibilty

We have been asked to develope a facebook application which decides friendship compatibilty between two users based on their first names on facebook. The core program behind this application needs to count number of character differences in the two given names.

Now, you have to write this small program to count number of character differences given two first names.

Faculty Code: CharDifference.java java (1.7)

```
import java.io.IOException;
import java.util.Scanner;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;
import java.util.TreeSet;

/**
 *
 * @author prakash
 */
public class CharDifference {

    static int difference(String string1, String string2) {
        int diff = 0;
```

Ln: 60, Ch: 1, | Total Ln: 61

[✓] Toggle editor Test

Input Specification: Two strings separated by a space

Output Specification: Number of character differences (not common charcters) as an integer

Please Provide Test Cases: 1 2 3 4 5 6 New test case

Testcase1:
```
jagdish tomy
```
Hint: Level:

Sample Input/Output: **Input**

Max Execution Time: 00 : 10 (min : sec)

Search Tags: character, string

Separate each tag with a comma (,)

Submit Cancel

*Figure 2: Teacher creates a question*

## Parikshak approach

Around this system, an approach to use it has evolved keeping in mind the issues mentioned above. Given the tendency of the students to directly start typing code in an IDE, when given a problem to solve, our approach enforces a ‘thinking’ or ‘writing’ phase where the student has only pen and paper, and is expected to study the problem in detail, create an approach, decide the key elements of the solution and build up test cases. Students who use this phase effectively have a higher chance of completing the problem well. Since IDEs provide shortcut to many tasks, often students lack an understanding of what is happening behind. I am reminded of one student who in an interview said “press F9”, when asked how do we compile a C program. So, we advocate working without IDE when learning to program.

To prevent students from altering the program according to specific inputs, Parikshak does not reveal the input cases used for testing the program. However, users are allowed to create their own test cases, and test their solution before submitting. This is meant to encourage careful planning and design of test cases when working on a problem.

As mentioned above, checking a program for correctness involves multiple aspects and is time consuming. Parikshak resorts to a test-case based comparison of outputs in this regard. Teachers create a set of test cases (5-8 per program is recommended) and load them in Parikshak (See figure 2). While preparing the test cases, teachers should minimise guessing opportunities, and ensure that as wide a variety is followed covering boundary cases also. Teachers are also required to write the correct or master program. Apart from providing a base for comparison, this will help teachers to get a realistic estimate of the time and effort required to build the solution.

Comparing outputs of two programs has its own challenges. Minor changes in output format – inserting a blank, putting a prompt, etc – can make simple comparison fail. A two-pronged approach is adopted to address this problem. On one side, additional filters are provided to make the comparison proper on a problem to problem basis. These include remove white space, trim white space, substring finding, etc. On the other side, we instruct teachers and students to stick to specification with respect to output.

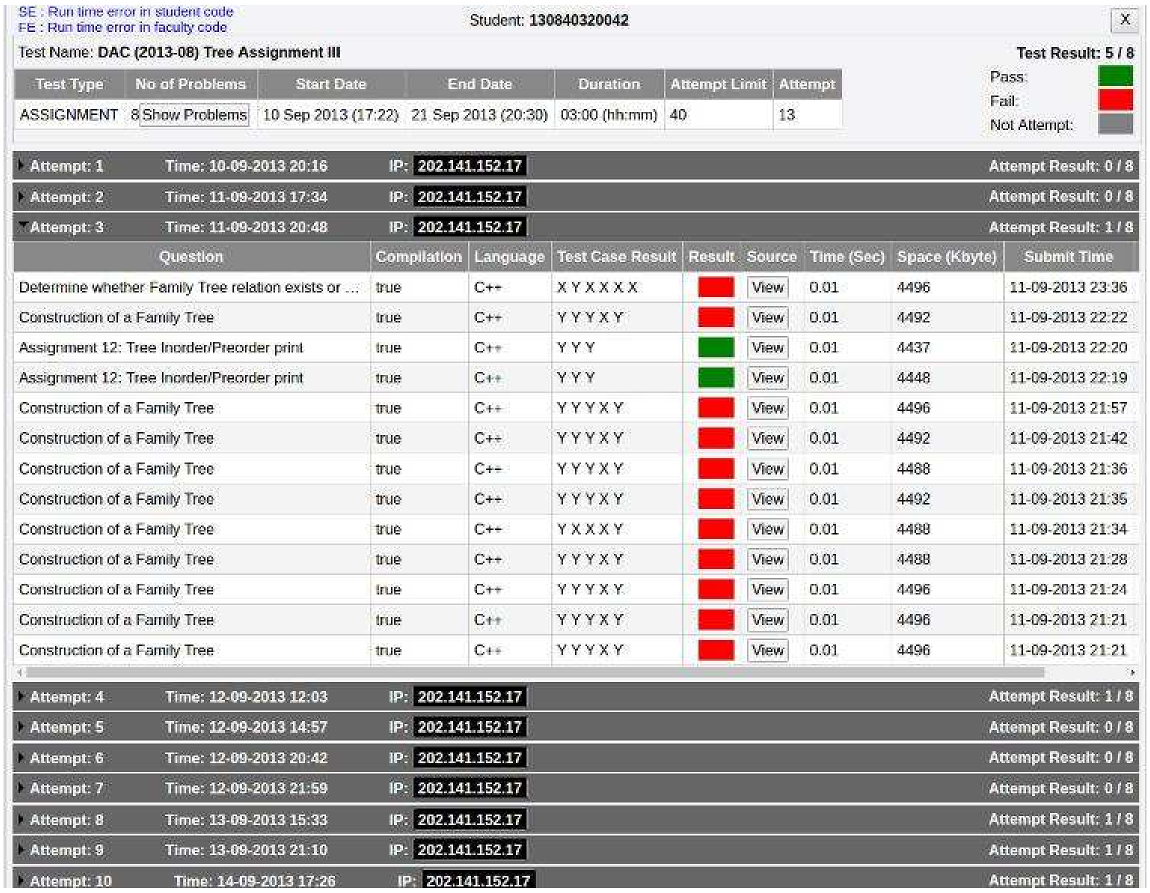
A bunch of such guidelines go to define the Parikshak approach. More details of this were presented in the T4E conference in 2010 at IIT Bombay [1].

Students find the Parikshak style challenging, primarily because they never are required to approach a problem from statement to complete working solution in one go. And usually debugging a program is a highly ignored component in labs. Students are often tempted to redo the program from scratch than debug a program. Putting time limits prevent this to some extent.

## Using Parikshak to address the problem

### Colleges

For colleges, Parikshak can provide an effective way to organise the programming labs. Parikshak portal already has a lot of programming problems; many of them are freely usable. (Teachers creating a problem have the option of making it open to all, or keeping it private). A set of problems can be selected at the beginning of the term, and enabled as an assignment or exam, for the specific students. Teachers can also create new problems as per the syllabus and add them. Students can now attempt them as per their choice, whenever and wherever they like.



SE : Run time error in student code
FE : Run time error in faculty code

Student: **130840320042**

Test Name: **DAC (2013-08) Tree Assignment III**

**Test Result: 5 / 8**

| Test Type | No of Problems | Start Date | End Date | Duration | Attempt Limit | Attempt |
|---|---|---|---|---|---|---|
| ASSIGNMENT | 8 Show Problems | 10 Sep 2013 (17:22) | 21 Sep 2013 (20:30) | 03:00 (hh:mm) | 40 | 13 |

| | |
|---|---|
| Pass: | 🟩 |
| Fail: | 🟥 |
| Not Attempt: | ▨ |

**Attempt: 1** **Time: 10-09-2013 20:16** **IP: 202.141.152.17** **Attempt Result: 0 / 8**

**Attempt: 2** **Time: 11-09-2013 17:34** **IP: 202.141.152.17** **Attempt Result: 0 / 8**

**Attempt: 3** **Time: 11-09-2013 20:48** **IP: 202.141.152.17** **Attempt Result: 1 / 8**

| Question | Compilation | Language | Test Case Result | Result | Source | Time (Sec) | Space (Kbyte) | Submit Time |
|---|---|---|---|---|---|---|---|---|
| Determine whether Family Tree relation exists or ... | true | C++ | X Y X X X X | 🟥 | View | 0.01 | 4496 | 11-09-2013 23:36 |
| Construction of a Family Tree | true | C++ | Y Y Y X Y | 🟥 | View | 0.01 | 4492 | 11-09-2013 22:22 |
| Assignment 12: Tree Inorder/Preorder print | true | C++ | Y Y Y | 🟩 | View | 0.01 | 4437 | 11-09-2013 22:20 |
| Assignment 12: Tree Inorder/Preorder print | true | C++ | Y Y Y | 🟩 | View | 0.01 | 4448 | 11-09-2013 22:19 |
| Construction of a Family Tree | true | C++ | Y Y Y X Y | 🟥 | View | 0.01 | 4496 | 11-09-2013 21:57 |
| Construction of a Family Tree | true | C++ | Y Y Y X Y | 🟥 | View | 0.01 | 4492 | 11-09-2013 21:42 |
| Construction of a Family Tree | true | C++ | Y Y Y X Y | 🟥 | View | 0.01 | 4488 | 11-09-2013 21:36 |
| Construction of a Family Tree | true | C++ | Y Y Y X Y | 🟥 | View | 0.01 | 4492 | 11-09-2013 21:35 |
| Construction of a Family Tree | true | C++ | Y X X X Y | 🟥 | View | 0.01 | 4488 | 11-09-2013 21:34 |
| Construction of a Family Tree | true | C++ | Y Y Y X Y | 🟥 | View | 0.01 | 4488 | 11-09-2013 21:28 |
| Construction of a Family Tree | true | C++ | Y Y Y X Y | 🟥 | View | 0.01 | 4496 | 11-09-2013 21:24 |
| Construction of a Family Tree | true | C++ | Y Y Y X Y | 🟥 | View | 0.01 | 4496 | 11-09-2013 21:21 |
| Construction of a Family Tree | true | C++ | Y Y Y X Y | 🟥 | View | 0.01 | 4496 | 11-09-2013 21:21 |

**Attempt: 4** **Time: 12-09-2013 12:03** **IP: 202.141.152.17** **Attempt Result: 1 / 8**

**Attempt: 5** **Time: 12-09-2013 14:57** **IP: 202.141.152.17** **Attempt Result: 0 / 8**

**Attempt: 6** **Time: 12-09-2013 20:42** **IP: 202.141.152.17** **Attempt Result: 0 / 8**

**Attempt: 7** **Time: 12-09-2013 21:59** **IP: 202.141.152.17** **Attempt Result: 0 / 8**

**Attempt: 8** **Time: 13-09-2013 15:33** **IP: 202.141.152.17** **Attempt Result: 1 / 8**

**Attempt: 9** **Time: 13-09-2013 21:10** **IP: 202.141.152.17** **Attempt Result: 1 / 8**

**Attempt: 10** **Time: 14-09-2013 17:26** **IP: 202.141.152.17** **Attempt Result: 1 / 8**

*Figure 3: Detailed student log for a teacher*

Teachers can monitor the activity online (while the lab is on) or offline, and find out what problems are being attempted mostly, or rarely, and common problems or mistakes people make. This can be used to plan effective discussions in the class, including specific feedback. At the end, the teacher can get a complete record of the student's activity in the term (See figure 3), and this can be used to decide the grades in a transparent way. Overall, this approach can significantly enhance the lab experience for the students and the teachers.

**Individual**

Currently Parikshak supports individuals only through an institution. However, work is on to allow individuals to register and practice programming. This will allow interested students to practice programming as problem solving, for a variety of problems. They can reuse code from session to session, access previous solutions, and get test results on their code, without depending on the teacher's presence.

**Going further**

As teachers, students, and employers in the space of IT, we are concerned with the quality of learning of students. Of particular concern, is the ability to build working programs effectively. We feel Parikshak or tools with similar functionality can go a long way in improving the "programming as problem solving" capability of our students. It is best to run your programming labs under the Parikshak environment as mentioned earlier. Parikshak is available at a nominal cost from CDAC Mumbai.

**References**

[1] A methodology for enhancing programming competence of students using Parikshak, Rane-Sharma, A.; Sharma, C.; Raman, R.K.V.S.; Sasikumar, M. International Conference on Technology for Education (T4E), 2010, July 2010.

**About the author:** Dr M Sasikumar is currently Director of CDAC Mumbai. He has been with CDAC since 1987 after completing his BTech from IIT Madras. He completed his Post-graduation from IISc, Bangalore and Doctorate from BITS, Pilani. His areas of interest are Artificial Intelligence, and Educational Technology. He has co-authored two books, one is with Prentice Hall on Parallel Computing. 6 students have completed their PhD under his guidance. He has initiated and nurtured a number of projects in the area of AI and E-learning during his career. He has over 100 publications so far. He is programme committee member in a number of international conferences, member of Board of Studies and advisory board for select institutions. He is fairly active in social media like Facebook, LinkedIn, and Research Gate..

## Zero-Carbon City

There are 10 principles that are proposed to create an eco-city. The developers of Dongtan have used them in order to create a carbon-free city.

- Revise land use priorities -- Create green, and safe mixed communities
- Revise transportation priorities -- Favour foot, bicycle, cart and transit over auto
- Restore damaged urban environments
- Create decent, safe, and economically mixed housing
- Nurture social justice and create improved opportunities
- Support local agriculture -- Create community gardens
- Promote recycling and resource conservation
- Work with businesses to support ecologically sound economic activity
- Promote voluntary simplicity
- Increase awareness of the local environment

*Source & Courtesy: https://en.m.wikipedia.org/wiki/Zero-carbon_city*