# Laboratory Experience For An Introductory Computer Science Course Oriented Towards Software Engineering

## Introduction

As a discipline, Computer Science has seen many dramatic changes in its brief history. Through new textbooks and an evolving curriculum, the content of the undergraduate Computer Science education has for the most part kept pace with these changes. But the pedagogy of talking at the student and then assigning him/her to write a program each week has hardly changed. Not only is this pedagogy out of date - it is profoundly wrong. It emphasizes individual skill in writing short programs from scratch in a dead language. This emphasis is the antithesis of that needed by a contemporary computing professional.

We believe that the practice of computing should be the main focus an undergraduate Computer Science education, especially in the first two years. We believe that this pedagogical shift has, by far, the highest leverage for improving Computer Science education. To achieve this goal, we have based the core sequence of our curriculum on a series of closed laboratory activities. The lecture content and materials will be supportive of the laboratories rather than the other way around.

We have begun development of an entirely new undergraduate Computer Science curriculum. The core courses of the new curriculum are designed to be more mathematically rigorous, more practice-oriented, more closely related to the real-world environment. We want the students to think and then to try what they have just thought out.

We began development of the first course (1CS: Introduction to Computer Science) of our new curriculum during Spring, '92 and offered it in both the Fall, '92 and Spring, '93 semester. This paper will present a brief overview of course content, course evaluation, and operational problems of our laboratory experience.

## Course Content

As part of our new curriculum, we have chosen C++ as the programming language for all of our courses. We established 1CS to cover basic C++ syntax, elementary programming skills, elementary software engineering skills, elementary computer science concepts, elementary object-oriented programming and design, and what we call "professional computer literacy" (such topics as window based interface, mail, ftp, news, etc.).

The 1CS course is designed with 2 1-hour lectures plus 1 2-hour laboratory per week. The lecture is given by a CS faculty member to all the students while the laboratory is divided into groups of 27-30 students per session with 4 Teaching Assistants supervising.

Each student has his/her own computer to work on.

## Laboratory Experience

The laboratory activity is designed to illustrate the concepts from lecture using examples, implementations, and problems for the student to work through. The activities allow the student to explore various ways to complete the problem. One of the goals of the activities is to provide an opportunity for the student to make mistakes, to try various options without serious consequences.

There are various checkpoints built into the laboratory activity where the student and the Teaching Assistant must talk about what has just occurred. The TA's function is to ask questions of the student about what he/she has just done - why he/she thought to do it this way, what other options did he/she consider and why were they ruled out. The TA's role is to facilitate thinking, not simply to mark a task completed.

In the real-world working environment, large projects are assigned to a team of people. Solo efforts are most likely small parts to a bigger group effort. Because of this, many of the laboratory activities are to be done with a partner or in a small group. By working with others, the student has an opportunity to experience a more real world work situation. The partners can ask each other questions, answer questions for others in the group, share the responsibility for teaching and learning.

This paper will detail two particular laboratories. The first of the laboratories focuses on an object-oriented programming concept, the use of structures in C++. This laboratory activity is based on the use of a particular data structure which enables the user to capture a combination of data items under a single name. It gives the student an opportunity to gain a better understanding of data types and further develop his/her programming skills.

The second laboratory to be presented introduces the student to a large software system. He/she will consider some of the practical issues involved in dealing with the design, implementation, and maintenance of a large software system.

## Structures Laboratory

This laboratory activity introduces the students to a new and difficult concept in object-oriented programming. A structure is a data type which enables the user to capture a combination of data items under a single name.

The activity begins very simply with a brief, but concise discussion of a *struct*. The student then executes a series of changes/solves a series of problems about *struct*s in an existing file. The first steps of the sequence are simple and straight-forward with each subsequent step more complicated and thought-provoking. The TA and student have a number of check-points where the work done is examined and discussed. The activity is designed such that it would be very difficult to continue to the next step if the student did not reasonably understand the previous one.

This laboratory activity received a major revision from the first semester to the second. Students in the first semester were very confused and lost; very few completed the activity during the lab period. Comments from the second semester indicated the activity was more successful. The students thought it was hard but worth the effort.

A comment from one of the undergraduate TAs who was a student in the course the first semester illustrates student reaction to this laboratory:

> "I liked the lab that they did this week. I remember last semester our lab for structures was the pizza lab. That lab was so complicated that only a few groups got more than a couple of the functions we were supposed to do done. This lab incorporated ideas they already had learned like the selection sort with the new concept of structures without being TOO challenging like last semester's lab.

..... overall I liked this lab better. The people I worked with learned a lot through the lab."

**Large Software Systems**

We have made the software engineering perspective one of the major goals of this course and of the new curriculum. It is important for students to be made aware of and to practice software engineering skills early on. Students are introduced to software engineering issues such as software design considerations, documentation, code reuse, and maintenance at every level of the curriculum.

The students use a 60,000 line program called "Fractinit" - a program which calculates and displays fractals graphically. Fractinit was written by Messrs. Bert Tyler, Tim Wegner, Mark Peterson and Pieter Branderhorst. Fractinit is freeware and the copyright is retained by the Stone Soup Group.

In preparation for this lab, students were asked to respond to questions about their ideas on large software systems. These questions were meant to make them aware of some of the important issues related to large software systems. For example, students were asked to respond to statements such as:

"The most important attribute of a program is correctness. As long as a program runs properly, nothing else really matters."

"As long as a program is properly commented, there is no need to create any other documentation concerning it's design and internal organization."

The laboratory activity is divided into four main areas:

1) playing with fractals - gives the student an opportunity to see how this piece of software works; what options it has; the different ways it can be used, etc.

2) building Fractinit - allows the student to put together from the source code an executable form of the software; the student will see that the source code is located in a number of separate source files and that a single source file would be unmanageable; the student will have the opportunity to observe the compiler going through Fractinit file-by-file with a separate .obj file created for each .c file that is compiled and as the last step, see the linker combine all the .obj files into a single FRACTINT.EXE .

3) working within Fractinit - forces the student to look more closely at the Fractinit source code; students are encouraged to work in groups of 3 to 5; each team picked a problem from a set of three seemingly simple straight-forward questions.

a) How/where would you correct a misspelled author's name on the credits screen?

b) Where are the statements which move the highlight box to the left when you hit the left arrow key?

c) What is the purpose of the global variable fudge which is defined in FRACTINT.C?

4) class discussion about experience of working with a large software system - this gave the students an opportunity to share their frustrating experience trying to answer seemingly simple questions when they have no documentation, no comments within the code,

no idea how everything fits together. This gave them a taste of the type of software they could easily encounter in the real world and the type of software they could be guilt of writing if they do not carefully plan out the project.

The homework assignment asked the student to go back to the prelab survey and discuss one of the statements, preferably one for which he/she would answer differently after this laboratory activity.

In the final course evaluation, this laboratory activity was mentioned as one of the most interesting, educational and fun.

**Course Evaluation**

We have completed two semesters of 1CS with much success. Our students indicated through their evaluations that they believe this course offered a fun way to learn computer science. They believe that working in groups added a dimension which will be useful to them in future courses as well as in the working world.

There were several criticisms of the course. The main concerns were the inadequate text book, the limited number of TAs during the laboratory sessions, and the pace of the course. We think we have solved one problem, and are working on the other two. We have chosen a new text book for the course. The problem with the previous text book was that it is a C book, not a C++ book (at the time there were no introductory C++ books available.) Hopefully, this new text will provide the C++ examples the students sorely missed.

The limited number of teaching assistants may never be resolved. Students enjoy having the one-to-one interaction with the TAs, and unless we provide a TA for each student, this problem will always be mentioned. We have combined or eliminated some of the checkpoints in the laboratory activity. Some of the checkpoints will now be evaluated by his/her lab partners, and others have been changed to be part of the post-laboratory homework. We do believe, however, that having a low student to teaching assistant ratio in the labs (28-30 students to 4 teaching assistants) is one of the keys to the success of the course.

It is important to note that students were extremely positive on the interaction they had with the TAs and the instructor. The relationship which developed was mentioned many times as the key in making this course worthwhile and a good learning experience.

We have also altered the pace of the course somewhat. Student comments complained of the slow pace at the beginning and the too fast pace at the end of the semester. Many students wanted more work on the concepts which were presented at the end of the course and suggested moving more quickly in order to gain the time for the more complicated concepts and skills. We plan to present the object-oriented paradigm much earlier in the course. We are not planning to add more material but to allow the student to have more time to "play" with the concepts and ideas.

We plan to continue to evaluate the laboratory activities throughout this school year. We are more aware of the potential problem areas and hope to gain the experience and information needed to improve these activities.

Examples of student comments from the final evaluation of the Spring, '93 semester:

[Overall, I think ...]

"... that this has been a great learning experience. I never had any programming in high school, and I hated TruBasic. C++ however was alot of fun and for some reason, I think I really understand the language and its usefulness. ..."

"... this class was very beneficial for me. I believe that my TA's in lab really wanted me to achieve and do well. For this reason, I had the incentive to believe that I could do well. The group atmosphere & cooperative learning made the class enjoyable."

## Conclusions and Summary

We believe our 1CS course is on the right track. Student response has been overwhelmingly positive. Students indicate that the laboratory experience is a major contributor to their understanding of the course material and is the "fun" part of the course. Another indicator of the positive effect of the laboratory portion of the course is the large number of students from the course who volunteer to be undergraduate TAs. We literally have more (competent and qualified) undergraduate TAs than we know what to do with!

We have many more laboratory ideas and activities which we would like to present. Unfortunately, space does not allow for anymore thoughts and examples of how our laboratory activities demonstrate computer science and software engineering concepts. We believe our 1CS course offers our students a solid, up-to-date, real-world introduction to computer science.

We are most willing to share our experiences and materials with any interested individuals and schools.