A Laboratory Computer Science Course for Liberal Arts Students

John E. Howland
Computer Science Department
Trinity University
715 Stadium Drive
San Antonio, Texas  78212-7200

Voice:  (210) 736-7480
Fax:  (210) 736-7477
Internet:  jhowland@ariel.cs.trinity.edu
Great Ideas in Computer Science

(Course title from Alan Biermann)

A breadth-first course which covers 13 topics in computer science in a lecture course, while a co-requisite laboratory course provides lab experience with 13 prepared experiments. The Scheme programming notation is used consistently in lecture and laboratory as a lingua franca for computer science. Students are not taught Scheme programming, but rather, learn just enough Scheme syntax and semantics to be able to read and understand programs written by others. Working Scheme models for each computer science topic are presented in the lecture course and are studied experimentally in the laboratory course. The development of this course and laboratory was funded by the Meadows Foundation and NSF grant DUE 9452050.
Course History

At Trinity there was a need for new laboratory science courses which students could use to partially satisfy the science requirement in each student's common curriculum.

A new introductory computer science laboratory course (and co-requisite lecture course) was developed. In the lab course students are gathered together in a workstation laboratory at the same time to work in pairs performing a prepared laboratory experiment.
Course Features

One feature of this course is that it covers a variety of computer science topics at about the rate of one new topic per week. Consequently, only an introduction to each topic can be presented during the three lectures on each topic. As a result, the course emphasizes a breadth of understanding at the expense of depth of understanding of any single topic.

Another feature of this course is that, while it uses the Scheme programming notation extensively to describe and model each topic, the course does not attempt to teach students to become effective programmers. Students are taught just enough Scheme syntax and semantics to be able to read and understand Scheme expressions and programs written by others. Such programs form the basis of models of various computational structures such as computer circuits, arithmetic units, data structures, processors, processes, etc. Since these models are expressed in Scheme, the models can be executed on a workstation and form the basis of laboratory experimentation in the course.
Lecture Topics

1. Introduction to reading the Scheme notation
2. Computer organization
3. Computer arithmetic
4. Computer circuits
5. Algorithms
6. Data structures
7. Programming methodology
8. Software engineering
9. Language translation
10. Program execution time
11. Computer networks
12. Parallel processing
13. Computability
14. Artificial intelligence
Laboratory Experiments

1. Getting Started with the Scheme Notation.

In this lab, students familiarize themselves with the lab workstations, editor and Scheme notation. No lab report is required for this lab session.

2. Using Computer Science Department Laboratory Facilities.

In this lab, students learn the format of a lab report and perform the first simple experiment which is to determine how fast the workstation can add numbers. Students are introduced to the problem of experimental sampling.


The purpose of this laboratory experiment is to determine the relative performance of different arithmetic types on a lab workstation.

4. Designing and Verifying a 4 bit Binary Adder.

The purpose of this laboratory is to build a working model of a 4 bit binary adder from modeled circuit elements and verify its correct operation.

5. Implementation of an Algorithm.

This experiment involves the experimental estimation of the time required to evaluate \( \text{fibonacci } 100 \) using a recursive implementation of the fibonacci function. This experiment requires an understanding of recursive and iterative implementations of the fibonacci function.
Laboratory Experiments

6. Choosing a Data Structure.

The purpose of this laboratory experiment is to examine and compare two implementations of an abstract data structure for a stack. The first implementation uses functions to implement a constructor, predicates and accessors for stacks. The second uses an object oriented approach to implement stacks. Most students are surprised by the outcome of this experiment since they find their conjecture incorrect.

7. Programming Methodology.

In this laboratory problem students are given three different implementations of a system for performing exact rational arithmetic. The implementation has been carefully designed and layered so that operations are separated from data representations by using abstract constructors and accessors. They are asked to predict relative performance of each system, comment on the quantitative aspects of each implementation, gather experimental data and draw conclusions in a written laboratory report.

8. Software Prototypes.

In this laboratory problem students work with a prototype implementation of rational arithmetic operations which allow rational numbers to be combined with other types of numbers in arithmetic expressions. You are asked to evaluate the performance of the prototype implementation and write laboratory report.
Laboratory Experiments


In this laboratory experiment, students are asked to design a recognizer for a syntactic element of the Scheme notation, given the BNF syntax description.


In this laboratory problem students analyze a function which generates recursive process then consider a recursive version of the same function which generates an iterative process and finally examine the compiler output and analyze the performance of both functions.


The purpose of this laboratory problem is to evaluate the performance of a network computation in which two networked workstations cooperate to perform a calculation in parallel.


In this laboratory problem students will try to discover the behavior of a rule system by tracing the execution of that rule based system as it interprets different rule sets.
Laboratory Hardware

The Scheme notation plays a fundamental role both in the exposition of computer science topics and in the laboratory experiments. Because of this, an effort was made to design a laboratory facility which could be used for lecture exposition as well as laboratory experimentation.

Grant proposals to the Meadows Foundation and the National Science Foundation (Grant # DUE-9452050) were prepared and both groups agreed to fund 50% of the cost of laboratory equipment. A vendor competition involving Apple, IBM, Sun, SGI and HP was designed which involved running certain Scheme based benchmarks and meeting a color graphics requirement of at least 8 bits per pixel on a display of at least 1024 by 768 pixels. Each vendor also had to meet other requirements involving memory capacity, UNIX operating system, disk capacity and networking.

HP won the vendor competition with 17 HP 712 machines of varying configuration. There are 15 HP 712/60 16M student workstations, 1 HP 712/60 32M instructor workstation and 1 HP 712/80 32M server machine providing passwords and home directories to each student workstation. HP also supplied an X windows based software package, SharedX, which allows the instructor machine to share any of its windows with any of the student lab workstations. In addition, SharedX also allows the instructor to turn over control of any of its windows to any of the student workstations. This means that students can, at the discretion of the instructor, provide responses to instructor queries which can be seen at all of the other student workstations. We are conducting a variety of experiments on how to use this facility in a lecture and lab environment.
### Laboratory Hardware

One additional requirement, to be used in the lecture course, was the ability for a lecturer to be able to use a machine for presentations and demonstrations and have these presentations be visible on the screens of each of the student workstations. We proposed that this could be accomplished either by using a video amplifier/switching system fed by the instructor's workstation monitor or a network based software video feed.

When used either as a lecture room or as a lab room we seat two students in front of each workstation. This limits class size to 30 students per section which is an appropriate maximum size for this kind of course.
Expository use of Scheme

A small subset of Scheme suffices when used as an expository notation in a course like this. Students first learn the rule for evaluating function expressions and then learn a few exceptions (special forms) to the general rule. Specifically,

define
quote
lambda
if
and
or
begin
let
Modeling Circuits

(define bit-or
  (lambda (x y)
    (if (or (= x 1) (= y 1))
      1
      0))
)

(define bit-and
  (lambda (x y)
    (if (and (= x 1) (= y 1))
      1
      0))
)

(define bit-not
  (lambda (x)
    (if (= x 0)
      1
      0)))
Modeling an Adder

(define bit-half-adder
  (lambda (a b)
    (bit-or
      (bit-and a (bit-not b))
      (bit-and b (bit-not a))))

(define bit-adder
  (lambda (a b cin)
    (let* ((t (bit-half-adder a b))
           (g (bit-and a b))
           (p (bit-and t cin))
           (list (bit-or g p)
                  (bit-half-adder t cin)))))

(define wire-output
  (lambda (pin-number outputs)
    (list-ref outputs pin-number)))
Modeling a 2 bit adder

(define 2-bit-adder
  (lambda (a1 a0 b1 b0)
    (let* ((t0 (bit-adder a0 b0 0))
           (t1 (bit-adder a1 b1 w 0))
           (list (wire-output 0 t0)))))
  (list (wire-output 0 t1)
         (wire-output 1 t1)
         (wire-output 1 t0)))))
Student Response

Generally, student response to this course has been enthusiastic. They are relieved that they can learn computer science concepts and have hands on experience conducting experiments of various types without learning to write their own programs. Even though learning to program is not one of the course goals, many of the students find that they can write their own simple programs and seem to be pleased with this knowledge even though most of them will never find an occasion to program computers later in life.
Distribution of Course Materials

Preliminary versions of the course notes and laboratory experiments are available via the department's web server,

http://www.cs.trinity.edu/About/The_Courses

Students can use a web browser such as NCSA Mosaic or Netscape to view these materials. The instructor can share a Mosaic window on his machine with each student workstation and use Mosaic as a presentation program. Expository Scheme can be copied from the web browser presentation window and pasted into an instructor's XTerm window running a Scheme interpreter which is also shared with the student workstations to provide interactive demonstrations of a working Scheme model.

The course materials are also freely available to anyone else via the internet. Other distribution of course materials are available by contacting the author.