Using J as an Expository Language in the Teaching of Computer Science to Liberal Arts Students

John E. Howland
Department of Computer Science
Trinity University
715 Stadium Drive
San Antonio, Texas 78212-7200
Voice: (210) 736-7480
Fax: (210) 736-7477
Internet: jhowland@ariel.cs.trinity.edu

June, 1996

Abstract

APL and J are seldom, if ever, used in the teaching of college or university courses. Recently, the author has developed a new laboratory based computer science course for liberal arts students in which students are introduced to 13 core computer science topics. Programming language is used in an expository fashion to describe each topic by building simple working models of each topic. These models are then used as the basis of laboratory experiments in a co-requisite laboratory course. Students are not taught programming in this course, but rather, are taught just enough of the syntax and semantics of the language to be able to read and understand the exposition and models. Initially, Scheme was used in the lecture notes and laboratory materials developed for this course. Recently, however, an experiment is under way to replace the use of Scheme in this course by J. The development of this course and laboratory was funded by the Meadows Foundation and NSF grant DUE 9452050.[1]

Subject Areas: Computer Science Education, J, Arts and Humanities, Exposition.
Keywords: computer science laboratory course, J, exposition.

[1]This paper appears in ACM Quote Quad, Volume 26, Number 4, Pages 55-62, June 1996. The paper was presented at the APL 96 Conference at Lancaster University, UK, July 29, 1996.

1 Introduction

At my university there was a need for new laboratory science courses which would partially satisfy a student's common curriculum requirement. Each student is required to complete two or three science courses. Two courses suffice when one of the two courses is a laboratory science course, so there is some incentive to take a laboratory course because the common curriculum requirement is reduced by one course. Traditionally, laboratory science courses had been selected from the natural science disciplines of biology, geology, physics or chemistry. The author developed a new introductory computer science laboratory course (and co-requisite lecture course) in which students are gathered together in a workstation laboratory at the same time to work in pairs performing a prepared laboratory experiment.

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 two or 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 J 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 J syntax and semantics to be able to read and understand J expressions and programs written by the instructor. 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 J, the models can be executed on a workstation and form the basis of laboratory experimentation in the course.

2 J as an Expository Language for Computer Science

The philosophical motivation for the choice of J as an expository notation for computer science is derived from its natural language sentence-like syntax [Kon 94, Rie 93]. Since the majority of the students enrolling in this course are likely to be fine arts and humanities students who are generally less oriented towards technology and science, it was thought to be useful to use a notation which could be easily related to the structure and form of natural language.

3 Lecture Topics

The lecture course consists of about three lectures on each of the following topics:

1. Introduction to reading the J 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

Covering this many topics in a three credit hour course requires adhering rather rigidly to a schedule. Fortunately, many of the topics are inter-related so that knowledge gained in one topic is immediately used in a following topic. For example, the logical organization of a computer is expanded on when discussing computer arithmetic. The modeling of computer circuits fills in some detail purposely left out of the computer organization topic.

This fast paced presentation of topics leaves little time for discussion of topics in the lecture setting. Some discussion of each topic is done during the weekly meetings of the laboratory course. Recently, an experiment involving required out of class discussion of topics on a local USENET discussion group for the class, has been started. Each student is required to contribute one new discussion thread each week and respond to two or three threads per week. The news group is not moderated, but is archived. Other members of the university community have been invited to read and/or join the discussion of these topics.

A set of lecture notes has been constructed which use J as an expository notation for computer science topics. Each topic has one or more J based descriptive models which is human readable by the students and executable by the laboratory workstation. These models form the basis of laboratory experimentation as well as providing an interactive working model of the topic under discussion so that students may obtain hands-on experience with each topic.

The lecture notes are available to students via a world wide web server and the instructor can use the WWW presentation during lecture presentations. A discussion of how this is accomplished is presented in Section 5.

4 Laboratory Experiments

The one credit laboratory course meets once each week for three hours. Students work in pairs to perform a prepared laboratory experiment. Each pair of lab students has its own laboratory workstation where the experimental work is performed. Currently, the laboratory course consists of twelve experiments. All experiments, except the first, require the students to formulate a conjecture to be verified, gather experiment data, analyze the results and write a brief laboratory report which must be turned in at the beginning of the next laboratory class period. With each offering of the course we have attempted to modify and improve the experiments as well as devise new experiments.

The following are representative titles and purpose of experiments used in the course.

- Getting Started with the J Notation.

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

- 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.

- How Fast Do Computers Perform Arithmetic.

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

- 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.

- Implementation of an Algorithm.

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

- 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.

- 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.

- 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. They are asked to evaluate the performance of the prototype implementation and write a laboratory report.

- Recognizing Syntactic Elements of a Language.

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

- Recursive Processes, Iterative Processes and Compiling.

  In this laboratory problem students analyze a function which is evaluated using both a recursive and an iterative process. Interpreted and compiled versions of each are analyzed.

- Evaluating Parallel Performance.

  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.

- Expert Systems: A Rule Interpreter.

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

This course is the first course in our computer science department which attempts to utilize the traditional closed science laboratory approach to teach computer science concepts. The effort required to design a successful laboratory experiment is much higher than anticipated, but the benefits can be rewarding in that successful experiments seem to convey the concept being taught more efficiently and forcefully than the more traditional kinds of programming problems we have assigned in other introductory courses. Our department plans to introduce closed laboratories for other introductory computer science courses as a result of our experience with this course.

5 Laboratory Hardware

The J 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.

For a number of reasons, including using this laboratory/teaching facility for a variety of other courses, we made a decision to base this laboratory on machines which run the UNIX operating system. J 2.06 is used on the lab machines.

Grant proposals to the Meadows Foundation and the National Science Foundation (grant DUE-9452050) were prepared. Each organization 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 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.

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.

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 login id's 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.

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.

6 Examples of J in Exposition

A small subset of J suffices when used as an expository notation in a course like this. In this section a few examples of the expository use of J are given. A few helping definitions are needed. Except for a few primitives such as addition, subtraction and catenation, English words are used in place of the primitive J

spellings to improve readability. Usually, the English word used is taken from the J Dictionary [Ive 95].

```
from =. {
amend =. }
take =. {.
drop =. }.
rep =. #:
base =. #.
time =. 6 !: 2
display =. 1 !: 2 & 2
format =. ":
```

The following words are defined for the primitive circuit gates.

```
or =. +.
and =. *.
not =. -.
```

The circuit elements and, or and not can be modeled as:

```
bitOr =. 3 : 0
('a' ; 'b') =. y.
a or b
)

bitAnd =. 3 : 0
('a' ; 'b') =. y.
a and b
)

bitNot =. not
```

A half-adder can be modeled using these elements as:

```
halfAdder =. 3 : 0
('a' ; 'b') =. y.
bitOr (bitAnd a , bitNot b) , bitAnd (bitNot a) , b
)
```

Next we can build a 1 bit adder using two half-adders as:

```
bitAdder =. 3 : 0
('a' ; 'b' ; 'cin') =. y.
t =. bitHalfAdder a , b
g =. bitAnd a , b
p =. bitAnd t , cin
(bitOr g , p) , bitHalfAdder t , cin
)
```

If a language feature is not in the student's current vocabulary when a particular model is described, then that feature would be introduced as a part of the explanation of the model. The idea is that certain language features are introduced when there is an expository need for them so that the focus is mainly on the topic being described rather than the programming notation. Also, notice that explicit definition, except in the simplest cases, rather than tacit definition is used because it is felt that students find it easier to understand explicit references to function arguments. The arguments passed to many of these functions are assigned, indirectly, to local names in an attempt to improve readability. Finally, the above models are monadic since bitAdder requires three arguments; the two summands and a carry input.

Next we can model a wire for connecting circuit elements as:

```
wireOutput =. 3 : 0
('pin' ; 'outputs') =. y.
pin from outputs
)
```

We can use two bit-adders and some wire to build a 2 bit adder as:

```
twoBitAdder =. 3 : 0
('a1' ; 'a0' ; 'b1' ; 'b0') =. y.
t0 =. bitAdder a0 , b0 , 0
t1 =. bitAdder a1 , b1 , wireOutput 0 ; t0
(wireOutput 0 1 ; t1) , wireOutput 1 ; t0
)
```

It needs to be emphasized that while students find such descriptions readable, they may also be used to provide an interactive working model which can be explored during a lecture using the classroom workstation and display system. These descriptions also constitute part of the prepared laboratory software for simple experiments involving computer circuit elements. In one laboratory experiment, students are asked to design and build a model of a 4-bit adder. The lab suggests that they verify experimentally that this adder performs 2's complement arithmetic and asks them to suggest a modification of their design which would allow the adder to subtract as well as add. The above models for adder circuits can be installed in a model of a simple CPU which is used in the lectures describing the organization of a simple computer.

In the lectures on software design and engineering principles, when discussing techniques of modularization and layering of software, the routines for rational arithmetic found in Structure and Interpretation of Computer Programs [Abel 85] have been used. One can demonstrate that it is possible to change the implementation of rational numbers without changing any code in layers above the implementation layer. The benefits of abstraction barriers is explored experimentally in a lab which asks students to form hypotheses concerning three different rational number implementations (don't remove common

factors from numerator and denominator, remove common factors at construction and remove common factors at access), gather data, analyze results and draw conclusions. This experimentation involves comparisons based on speed or space used.

In the lectures on computer organization, a very simple one accumulator computer, having four instructions (load, store, add and subtract) is described. A J working model of this computer is given which allows students to not only see, in precise terms, the organization of this machine, but also run a few simple programs on this machine.

The memory of this machine is modeled as a J vector. For example,

```
mem =: 0 0 0 0 307 108 409 22 3 0
```

models a memory containing a machine language program, starting in location 4, for the following J expression.

```
c =: a + b
```

A memory system has two operations, access and store. These are modeled in J as:

```
access =. 3 : 0
y. from mem
)

store =. 3 : 0
mem =: (1 from y.) (0 from y.) amend mem
)
```

A processor is modeled by a J function named proc. The processor contains three registers which are modeled by global variables *ac*, the accumulator, *pc*, the program counter and *ir*, the instruction register. The processor is modeled in J as:

```
proc =. 3 : 0
whilst. y. do.
     NB. show registers and memory if tracing
     displayRegisters ''
     NB. fetch instruction
     ir =: access pc
     NB. increment pc
     pc =: pc + 1
     NB. decode instruction
     ('opCode' ; 'address') =. 100 100 rep ir
     NB. interpret instruction
     if. opCode = 1
          do. ac =: ac + access address continue. end.
```

```
        if. opCode = 2
             do. ac =: ac - access address continue. end.
        if. opCode = 3
             do. ac =: access address continue. end.
        if. opCode = 4
             do. store address , ac continue.
             else. 'invalid operation code' break. end.
    end.
    )
```

The processor is started with an argument of zero to execute a single instruction at a time, while running the processor with an argument of 1 causes continuous operation of the machine. A global variable, trace controls whether or not the registers and memory are displayed before each machine cycle as shown in the displayRegisters function.

```
displayRegisters =. 3 : 0
if. trace
     do.
          display 'pc= ' , (format pc) , ', ir= ' , (format ir) , ', ac= ' , format ac
          display 'mem= ' , format mem
     else.   0
     end.
)
```

Here, again, the point of such descriptions is that they are readable by humans, yet each such description is also a working model which can be explored in the laboratory.

7 Student Response to the Course

Generally, student response to this course has been enthusiastic. They seem to be 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. Since teaching programming is not the focus of this course, students tend to be casual rather than expert readers of the programs used in this course. J's readability could be improved for this type of user. For example, one nice feature of J is that one can use alternate spellings for words. This is useful when natural language readability is preferred over the more terse J spelling, for example, using the word *or* rather than *+.*.

However, some words have an interaction side effect which preclude use of an alternate spelling such as *define* for *3 : 0*.

Also, casual readers of the language seem to prefer arbitrary pronouns to name function arguments rather than the fixed pronouns *x.* and *y.*.

An example of this style of expression can be found in the definitions of many of the functions given in Section 6. J users may find it preferable to have a definition syntax which would allow indirect specification of argument names *a*, *b* and *cin* in a more convenient form than *('a' ; 'b' ; 'cin') =. y.*.

Each offering of this course causes a few students to consider majoring in computer science. Hopefully some of those students will have learned enough J to realize the advantages of continued use of a language such as J.

The first few offerings of this course occurred before the acquisition of the laboratory equipment described in Section 5. The course was run in a makeshift lab of borrowed, out-of-date, Sun and Apple UNIX workstations. The lab workstations were under-powered for some of the experiments and presented different user interfaces at student workstations. Student response to laboratory experiments is much more positive with the HP workstations.

8 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 J can be copied from the web browser presentation window and pasted into an instructor's Xterm window running a J interpreter which is also shared with the student workstations to provide interactive demonstrations of a working J model during lecture presentations.

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

References

[Abel 85] Abelson, Harold and Sussman, Gerald with Sussman, Julie., *Structure and Interpretation of Computer Programs*, MIT Press, 1985.

[Bla 76] Blaauw, Gerrit, *Digital System Implementation*, Prentice-Hall, Inc., 1976.

[Ive 95] Iverson, Kenneth E., *J Dictionary*, Iverson Software, 1995.

[Kon 74] Konstam, Aaron and Howland, John, "APL as a Lingua Franca in the Computer Science Curriculum", SIGCSE Bulletin, Volume 6, Number 1, February 1974.

[Kon 94] Konstam, Aaron and Howland, John, "Teaching Computer Science Principles to Liberal Arts Students Using Scheme", SIGCSE Bulletin, Volume 26, Number 4, December 1994.

[How 95] Howland, John, "A Laboratory Computer Science Course for Liberal Arts Students", The Journal of Computing in Small Colleges, Volume 10, Number 5, May 1995.

[Rie 93] Riehl, Arthur, moderator, "Using Scheme in the Introductory Computer Science Curriculum", Panel, SIGCSE Bulletin, Volume 25, Number 1, March 1993.