CSc 451, Spring 2003
Project Assignment
Due: Wednesday, May 7 at 23:59:59

The project assignment is simple in essence: Develop an Icon application of significant size and convince me that it works. The application may be a tool, a game, or a visually pleasing amusement, to name three possibilities among many.

Selection of a Project

There are two keys to success with a wide-open project of this sort: (1) Find a problem that interests you. (2) Find a problem with a clearly attainable nucleus that can be gradually built upon.

It's well known that people do their best work when they're working on something that they find interesting and are enthusiastic about. Maybe there's a tool that you've always wished for—now you can build it! If you like computer games, maybe a game would be a good choice for you. If you're artistically inclined you might produce a graphical program that simply produces appealing abstract forms. If you're pursuing a double major or a minor, that other field of study might be fertile ground for finding a good project. If you're interested in the Web, you can certainly use Icon for CGI scripts.

Regarding the second key to success, to avoid a failing grade it is imperative that your project works to some significant extent. <u>There is great danger in selecting a project that has a key central problem that you are not sure you can solve.</u> Consider handwriting recognition for example: If in the end the program can't really recognize much of anything, that would likely result in a failing grade for the project.

As a good example of a project with an attainable nucleus that can be built upon, consider a text editor. In the very simplest case an editor need only be able to read a file, write a file, list the current set of lines, and allow the user to add or delete lines. That would be about a 10-point homework assignment but features can easily be added to make it sufficiently challenging to be an acceptable project. If a planned feature turns into trouble, it's not hard to find a feature to substitute.

As a rule, no two projects may be the same but a reason to make an exception on this rule may emerge. By default, however, if there is a collision the first proposal wins.

Another issue to consider when considering a potential project idea is the technology you'll be counting on. I think of the core Icon implementation as being extremely solid. I'll be very surprised if anyone encounters a bug of any significance in the language itself. Icon graphics facilities are also well-proven and stable. Unicon is a work in progress, however. If you take a look at the `unicon-group` mailing list archives you'll see that bugs in Unicon do turn up from time to time. The core elements of Unicon are reasonably stable but as you move away from that core, perhaps towards packages described in the Unicon book, you may find a few bugs.

Project Ideas

Here are some very sketchy ideas that I think have the potential to be the basis of a project:

- A graphical editor for something like UML.
- An animation of program execution in an interesting language, such as Prolog
- Type inferencing for a language such as Icon. Maybe use a Lisp-like representation to focus on semantics instead of syntax.
- Read English text and produce a "translated" version that's comically garbled, like the current Sprint commercials. "Soundex" coding would be the place to start.
- A typing tutor.
- Windows registry browser—work with a text-based dump of the data.
- Add features to your favorite shell via a preprocessor.
- Look for cheating in programming assignments. For a start on this see `dv.icn` in Spring 1997 372 assignment #6.
- A browser for Java classes or XML or files, etc. There are lots of possibilities with additional ways to view data.
- A tool to help instructors learn the names of students in a large class.
- Calendar/reminder tool
- Take a collection of mail messages and extract a relatively minimal dialogue.
- Meeting scheduler
- An Icon debugger
- A program to help would-be artists understand drawing with perspective.
- A manipulative puzzle such as Rubik's cube, although Rubik's cube is specifically not allowed.
- A program that would allow an instructor to browse `turnin` submissions. You can find a tiny of bit this in the Spring 1996 CS 372 assignment 4: `status.icn`.
- Continue on the theme of `mtimes.icn`. I picture a graphical tool that is perhaps generalized to handle similar data from other domains.
- Transform program source code in some way. For example, read a Java source file

and insert print statements that produce call-tracing information.

Here are some ideas from Ralph Griswold:

- A cryptographic workbench: Cryptography in the classical sense—substitution and transposition ciphers, tools for decrypting, etc.

- English-language text synthesis: How about a program to write short stories, government gobbledegook, poetry, or what not.

- Document formatter. Design would be challenging but it is an embellisher's dream.

- Stylistic analysis of natural-language text to ascertain, for example, its authorship. You should know something about linguistics to try this one.

Two potential project areas comes to mind that do not involve a large application:

(1) Do performance analysis of Icon versus another language such as Java, Perl, Python, etc.

(2) Develop a library of Icon procedures for a particular application area.

We are just now moving into graphics but you can certainly thumb through *Graphics Programming in Icon* to get an idea of capabilities. Ditto for Unicon.

Steps to Success

The first step is to identify one or more potential projects that you have an interest in. For each potential project write a paragraph or two that outlines the basic concept and the key functional aspects of the application you have in mind. Send that to me via e-mail or discuss it with me in person. I'll make an initial assessment of whether an idea is a good project candidate. You should aim to provide me with some ideas to consider no later than Monday, March 24, and hopefully much sooner.

Given a project candidate the next step is to reach agreement with me on a sufficiently challenging set of features. You'll need to present me with a written list of features that you believe is sufficient. A sentence or two per feature is probably enough for me to get an idea of what you've got in mind but I may ask you to follow up with more detail on some items. I will negotiate with you to find some middle ground where I believe the feature set is sufficiently challenging and you believe you can get it done with a reasonable amount of work.

Homework assignments and the project are each about one third of the grade so one way to think about the size of the project is that it should take about as much effort as all the homework assignments put together. Lines of code is not a very good measure but I anticipate that projects will typically run between 1200 and 1800 lines of well-written code.

The end result of the negotiation will be an Agreed Feature Set (AFS). When the project is graded, if all the features on the AFS are present and work well, if the supplied documentation is adequate for me to successfully use the program, and if good use is made of the language and facilities, that will be worth an "A+".

You should aim to have an Agreed Feature Set in hand no later than Thursday, April 10. It will take the form of a printed copy of your feature list that bears my signature.

If you're familiar with the idea of Use Cases and/or User Stories, your AFS may employ those concepts.

Writing software is an inherently unpredictable proposition and it may turn out that as you begin implementing your project you find that your Agreed Feature Set is overly optimistic. If so, you may try to renegotiate the AFS but to be successful with that you'll need to make a convincing argument. Similarly, if you determine that a lesser grade on the project, such as a "B", will meet your academic objectives, you may negotiate an AFS reduction that would be still be sufficient to yield a "B".

In some cases I may agree to an AFS that I later come to view as insufficiently challenging. In such a case I may appeal to the student's sense of dignity, decency and self-respect to encourage that student to do a little more than we agreed upon, but in the worst case I will honor the AFS.

NOTE: It is important to recognize that all projects will not be equally difficult. With a typical homework assignment everybody solves exactly the same problem and the result is completely equitable. For a project of this sort everybody will be solving a different problem and although I'll do my best to make sure that the AFS for each project is reasonably challenging, it is virtually impossible to ensure that all projects are equally challenging.

Grading

The project constitutes thirty percent of the grade for the course. For grading, a project can be evaluated in several dimensions, including:

- Overall functionality
- User interface design
- Robustness
- Code quality
- Accompanying documentation

For example, it is reasonable to expect that an ambitious project with strong functionality may be less robust than a smaller, simpler project. A graphical user interface is more difficult to develop but typically requires less documentation than a line-oriented interface. Some applications may require virtually no documentation.

Because of the potential variety of projects, and rather than having the project score be comprised of the sum of scores in several areas, a single grade will be assigned to the project as a whole. In turn, that grade will map to a percentage score based on the following table:

A+	100
A	95
A/B	90
B	85
B/C	80
C	75
C/D	70
D	60
D/F	50
F	0

The one quality that is required of every project is good use of the Icon programming language. Icon's types, operations, and goal-directed evaluation model should be taken advantage of wheneverapplicable.

Deliverables

Use the `turnin` tag `451_project` to submit your project for grading. Only two files have required names: `Readme` and `Guide.txt`. The `Readme` file provides a "roadmap" to everything else. The `Readme` file should:

(1) Contain an overview that describes the basic idea of the project and provides a high-level feature list.

(2) Describe how to build the project from source files. Arrange things so that only one simple step is required to produce an executable. You might use a Makefile or just supply a shell script.

You may use whatever names you desire for source files, data files, test data, etc., but be sure to turn in everything that's needed to build and run the program. Note that a directory name can be supplied to `turnin`, causing all files in the directory to be turned in.

Documentation for the project should be in `Guide.txt` if plain text, such as created with `pico` or `Notepad`, or in `Guide.ps`, `Guide.pdf`, or `Guide.html` for PostScript, PDF, or HTML formats, respectively. DO NOT submit multiple `Guide.*` files—pick one format.

The essential purpose of `Guide.txt` is to enable me to convince myself that your project fulfills the AFS. You might do that by providing a tutorial that covers all the features on the AFS. You might simply augment the AFS with descriptions of how to demonstrate each feature.

If your project is a non-interactive program such as a translator of some sort, you'll need to provide a collection of data files and instructions on how to use them to see the program in action.

Miscellaneous

Late projects will be accepted but with a penalty of 10 points (out of 100) per day, up to the day of the final exam. Submitting more than one day late may lead to a grade of "I" (incomplete).

You may incorporate existing public domain code, such as that in the Icon Program Library, into your project, or even use a program in the library as a starting point for your project, but you must include comments that clearly cite the source for any such material. If you anticipate using a significant amount[1] of existing code then you should bring that up as part of the AFS negotiation. If an AFS has been established and you then discover a significant amount of existing code you wish to use then you'll need to renegotiate the AFS with me.

There are lot of significant Icon programs floating around the net but I hope that no one will be tempted to plagiarize. The penalty for any plagiarism on the project will be a failing grade in the course.

---

[1] Consider it to be a significant amount if the total amount of existing code used exceeds 5% of the non-comment source code of your project. Library routines in `/home/cs451/lib` do not count.