Homework 0: Why?

Due: problem 1, August 28, 1997; problem 2 September 4, 1997; problem 3 in class; problem 4 September 16, 1997, problem 5 September 9, 1997.

The purpose of this homework is to give you and the rest of the class some background for negotiating about the syllabus. In addition, I hope this will increase your understanding of the goals and methods of programming languages, spark your interest in the subject, and increase your critical thinking skills.

Problems 2, 4, and 5 are to de done in groups, the others are individual problems. For the group problems, follow the course policies, in particular, be sure to have a leader, write out the certification required, and hand in a list of what worked well and what didn't.

Readings: See the books on reserve for the course, especially general books about programming languages, and the *Introduction to the Literature* handout.

1. (20 points) Read Hoare's [7] and Wirth's [10] articles on programming language design. Think about each one in relation to some programming language you know well, for example, C or C++. On paper, note what language you were thinking about, and write down two lists, or paragraphs, about how that language (a) does and (b) does not follow the ideas given in these articles. We'll be discussing these in class, so have these ready to turn in at the beginning of class.

2. (150 points) Choose *one* of the first following parts to do as your topic. Each is worth the total points. Your group will prepare a WWW page on your topic, and will also make a 20-30 minute presentation in class on the topic (more time can be used if needed). The web page will become part of the course's web site, so write it with the understanding that everyone will be able to see it. Include your names on the web page. You will make a "dry run" of your presentation with me before the actual presentation.

   I'll be evaluating this problem based on how "critical" your work is. That is, I want you to think clearly, based on logic, experiment, or evidence (if it exists), not just jump to conclusions. Consider the alternative positions seriously. Let someone in the group play "devil's advocate". Don't tell me what you think I want to hear; try to find the truth. For questions that might be answered by evidence, do some research to find what evidence there is; if you find evidence, cite it. If there isn't evidence to answer a question (and for some there isn't), be sure to say that what you're presenting is based on educated guesses, and say why you think it's reasonable. For questions you can answer by simple experiment, do the experiment; if the experiment is complex, look to see if someone else has done it (if it seems likely). For questions you can answer by logic, reason it out carefully. If you get material off the web, be sure to evaluate who is saying what, and what their motivations and qualifications are. I don't mean to have you go to excess here — don't do a master's thesis for each question — but do try to be critical within the constraints of the time you have.

   The style of oral presentation is up to you; you can have one person talk, or you can all present parts of it. If you need viewgraphs made, bring them to me early on the day of your presentation.

   (a) **History of Programming Languages.** You are stuck next to a recent graduate from high school on a long plane trip. This very talkative person has learned Java and thinks it's the only programming language worth knowing, but has a vague recollection of hearing about something called C++. You are appalled. You point out to this kid that he has no sense of the history of programming languages. The kid says "Oh yeah? Well, like, then, tell me about the history of programming languages." You recall a presentation you once made in Com S 541 about that very topic. . ..

   In your presentation, include the following aspects.

- What problem gave birth to programming languages?
- A time-line showing when various "important" languages were first defined. (Look in [9], for example.)
- What niche, application area, or clientele, each was intended for.
- A time-line showing roughly what the most widely-used or "hot" language was at any given time. Is there any periodicity in this time-line?
- Is there some factor that makes a language "hot"?
- The problems each language tried to solve, or the innovations that it made. For example, what languages responded to the "structured programming" crisis and how?
- Has the declining cost and speed of hardware influenced programming languages throughout history? Or are there other trends you can observe?
- What do you think of the 90% theory advocated by Gabriel [6]?

Although not necessary, keep your eye out for whether ideas migrate from "research" languages into widely used languages (or vice versa). Any other historical information that you think would be relevant can also be described.

(b) **Comparing Programming Languages.** You have just been promoted to management in the "itty bitty machines" company, a leading software house. Your first assignment is to choose a language for the next big development project for your company. The president of the company has a reputation for requiring critical analysis of such decisions, and so your boss wants you to first make a presentation to him about the basis for making such decisions. The topic, in short is: how does one compare programming languages?

Include the following aspects in your discussion, and give an example comparison to illustrate your description.

- What is "Flon's Axiom" [3] and what does it say about evaluating programming languages? Do you agree with it?
- What is the "Turing Tarpit" [5, 8] and what does it say about comparing programming languages?
- Is there any way out of the "Turing Tarpit"?
- Is there a distinction between a language and a compiler for that language? What about tool support (such as development environments) and documentation?
- Should one evaluate languages differently depending on their intended purpose?
- What criteria do you think are important in comparing or evaluating general purpose programming languages? Why are the criteria important?
- Do these criteria have any relevance for comparing things like OO class libraries, or different designs for an abstract data type?

As a bonus, if you have time or interest, you can describe theoretical work on comparing the expressiveness of programming languages.

(c) **Economic Impacts of Programming Languages.** You are a top researcher for "mega soft corporation." The president of the company, Buck Doorman, has grown tired over the endless changes of programming languages he's seen in the development laboratories, and of their endless squabblings over whether various languages are better than others. "We're a business" he says, "not a university." So he directs you to investigate economic impacts of programming languages. "I'm so tired of this," he says, "that I wonder if there's *any* difference at all between these languages."

In your presentation, include the following topics.

- Is there any data that show a difference in cost between using different programming languages? (See, for example software cost estimation models [2], and Boehm's experiment [1] and look for newer references to the latter using the science citation index.)
- What are the main costs affecting software production today?

- How does the choice of programming language or the aspects of a language affect these? (Perhaps it may help to look at broad categories of languages, such as functional and OO languages, and to consider extremes, such as machine language.)
- Does tool support make any difference? If so, how does it compare with the effects of the language itself?
- What tradeoffs can one make among costs when choosing a language for a development project?
- How does reuse of software affect the economics of software?
- What kinds of programming languages, or features, promote reuse?

As a bonus, describe what experiments could be done to improve our knowledge in this area.

(d) **Programming Paradigms.**

- What is a paradigm [4]?
- What is the difference between a paradigm and a programming language, if any? If there is a difference, what is their relationship? If there is a difference, can one program in any given paradigm in any given language? Is there any effect of the language?
- What is the difference between a paradigm and a style of software architecture, if any?
- Give a diagram that shows the major paradigms of programming, and perhaps relates these to programming languages.
- Is there any evidence to support the notion that it's a good idea to learn more than one paradigm of programming?
- What are the claimed advantages of each paradigm? (Give examples with code if possible.)
- What are the limitations or disadvantages of each?
- Is it possible to combine paradigms, or do they only exist in pure forms?
- What is a multi-paradigm language?

As a bonus, think about what other paradigms might be possible.

3. (30 points; individual problem) During the presentations of each of the other groups, write up a list of questions for them. Highlight especially points that they made that you think are not well supported or that don't show sufficiently critical thinking. You will hand these in at the end of their presentation.

4. (60 points) Respond to the questions raised by the other students in the class, in the previous problem, by listing the questions you think are significant, and your answers to them. To make this easier, you can just list for me:

   (a) the questions you think sound "significant" and your reasons why no or minimal response was needed, and

   (b) the questions you thought were "significant" and that you took into account on your web page

   along with any questions and answers deserve more discussion, but that aren't appropriate to put on the page. The first two lists can just be lists of questions in those categories.

   Hand in these, along with the group leader's statement and a list of what worked and didn't work for the group, and also send me your revised web page (or a pointer to the files).

5. (100 points) Prepare a list of objectives and a syllabus for the course, and make a printout of that. Use the format found on the course web page. Feel free to copy objectives from previous offerings of the course, provided that's what you really want. It's important to justify your objectives on paper. We will discuss this in class.

References

[1] B. W. Boehm. An experiment in small-scale application software engineering. *IEEE Transactions on Software Engineering*, SE-7(5):482–493, Sept. 1981.

[2] B. W. Boehm. *Software Engineering Economics*. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1981.

[3] L. Flon. On research in structured programming. *ACM SIGPLAN Notices*, 10(10):16–17, Oct. 1975.

[4] R. W. Floyd. The paradigms of programming. *Commun. ACM*, 22(8):455–460, Aug. 1979.

[5] J. Frederick P. Brooks. *The Mythical Man-Month*. Addison-Wesley Publishing Co., Reading, Mass., 1975.

[6] R. P. Gabriel. The end of history and the last programming language. *Journal of Object-Oriented Programming*, 6(4):90–94, July 1993.

[7] C. A. R. Hoare. Hints on programming language design. In A. I. Wasserman, editor, *Tutorial Programming Language Design*, pages 43–52. IEEE, Oct. 1980.

[8] B. J. MacLennan. *Principles of Programming Languages*. Holt, Rinehart and Winston, New York, N.Y., second edition, 1987.

[9] R. W. Sebesta. *Concepts of Programming Languages*. Benjamin/Cummings, Redwood City, Calif., third edition, 1996.

[10] N. Wirth. On the design of programming languages. In *Information Processing 74*, pages 386–393, New York, N.Y., 1974. North-Holland Publishing Co.