J. Nathan Foster: Teaching Statement

My motivation for pursuing a career in academia is rooted in a desire to teach. I care deeply about education and believe that teaching offers a unique opportunity for service—giving something back to society by helping train the next generation of scientists. I also find that teaching improves my own work—leading courses forces me to sharpen my knowledge of a subject, and working with enthusiastic students on research provides momentum that drives projects forward. I look forward to introducing students to the beautiful results that first sparked my interest in computer science, training them up with technical skills that will make them experts, and collaborating with them on research as an encouraging and nurturing mentor.

Whenever possible, I have taken advantage of opportunities to teach. At Penn, I worked as a teaching assistant for the graduate programming languages (CIS 500) and undergraduate data structures (CSE 121) courses. I specifically sought out these assignments because, along with the usual duties of developing and grading assignments, they offered the opportunity to lead my own recitation sections. I enjoyed having responsibility for my own group of students and gained valuable experience by presenting material to them in a classroom setting.

In the programming languages course, many of my students came from more practical backgrounds and were unfamiliar with the theoretical concepts we covered in the course: lambda calculus, type systems, induction, formal proofs, etc. After watching them struggle through the first few assignments, I decided to structure my recitations as hands-on problem solving sessions. In a typical recitation, I would briefly answer questions about the previous lecture and then spend the rest of the time working through proofs on the chalkboard or discussing general problem-solving techniques. Although my recitations were large—I had about 30 students in one—I did my best to get every student involved and keep discussion interactive by dividing them into groups to work on problems, asking them to come up and present solutions on the chalkboard, and frequently stopping my own lectures to ask questions. My students told me they found these sessions helpful, and they continued to attend my recitations in large numbers throughout the semester.

In the data structures course, the topic that my students found most challenging was complexity analysis. I devoted several labs to helping them get comfortable solving these problems, using many of the same techniques discussed above to encourage active participation. Another topic we devoted a lot of time to was debugging. The students were pretty good basic programmers, having just completed the introductory course, but they often got stuck when their code did not work as they expected. I spent one lab showing them how to use an interactive debugger and a lot of time over the semester discussing techniques for instrumenting code with assertions and debugging statements. I also designed the final course project, which had them implement a domain name server. In the project description, I gave a complete specification of the server, a set of sample client interactions, and some skeleton code for handling I/O, but I left most of the major design choices open, including the choice of the data structure for lookup tables. As this was the most freedom they had ever had on a project, I included a mandatory design phase: students submitted detailed design documents, which I read and returned to them with comments. To my great delight, almost every student in the course was able to complete the project and some of the students even started an informal competition to see whose server could answer queries fastest. I was awarded the departmental teaching award for my work in this course.

As an undergraduate at Williams, I worked as a teaching assistant for the introductory (CSCI 134) and programming languages (CSCI 334) courses, as well as a course on classical and quantum information theory (MATH/PHYS 316) in the math department. Later, at Cambridge, I gave "supervisions" in logic and verification (CST Specification & Verification) and advanced compilers (CST Optimising Compilers). Compared to my subsequent teaching experiences at Penn, my responsibilities in these courses were somewhat limited: at Williams I graded, assisted with labs, and held office hours, while at Cambridge I led tutorial-style meetings with two to three students. Even so, these experiences were valuable. I enjoyed helping students and I found that teaching forced me to master the material at a deeper level than I had as a student.

I have also worked with a number of undergraduate students on research projects. Even the brightest students

often find the transition from taught courses to research challenging. To help ease this transition, I usually start students with a small, implementation-oriented project. I find that this helps them to get involved quickly while keeping them from getting lost or bogged down in details. After they complete this initial project, I then regroup and help them pick a more substantial problem to work on. I strive to be a good mentor, providing encouragement and stepping in to help when things gets stuck, but also knowing when to step back so they have a sense of ownership of their work. Many of my students have achieved success, but two especially stand out: Ravi Chugh's senior design project on reversible string transformations won the Hugo Otto Wolf Prize from Penn, while Alexandre Pilkiewicz's work on Boomerang during his internship won the Grand Prix d'Option from École Polytechnique and led to coauthorships on publications at two top conferences.

In the future, I look forward to teaching any of the introductory courses in the undergraduate curriculum as well as advanced undergraduate and graduate courses on programming languages, compilers, databases, logic, discrete math, and theory of computation. In designing these courses, I would seek to develop assignments that would be fun for students to work on while still testing their mastery of the core material. I believe that projects are a good way to achieve both of these goals—they give students a chance to be ambitious and let them integrate skills from several different areas. Projects can be successful at all levels of the curriculum, although projects in introductory courses require more planning to ensure success. I am also interested in leading graduate seminars on advanced topics in programming languages, databases, security, and software engineering, perhaps in cooperation with experts in those fields. I am particularly interested in seminars on security-oriented languages, the logical foundations of regular types, contracts and hybrid type systems, implementation techniques for functional languages, model transformations, and data replication. One way to structure these seminars would be to start with a few classic papers to frame the topic and establish basic terminology, and then proceed to work from the recent literature.