Foundations of Programming

Bulletin Description

A first formal course in computer programming required (e.g. Comp 110, UNC). Advanced programming: objects, pointers, classes, interfaces, packages, inheritance, delegation, observers, MVC (model view controller), exceptions, assertions.

General Course Info

Term: Fall 2013
Department: COMP
Course Number: 401
Section Number: 001

Time: TR 11:00 – 12:15
Location: Room MY G202
Website: http://www.cs.unc.edu/~dewan/comp114/current/

Instructor Info

Name: Prof. Prasun Dewan
Office: FB150
Email: dewan@cs.unc.edu
Phone: 9621823
Web: http://www.cs.unc.edu/~dewan
Office Hours: TR 15:30 – 16:30

Teaching Assistants (TBA)

<table>
<thead>
<tr>
<th>Name</th>
<th>Office</th>
<th>Email</th>
<th>Office Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Textbooks and Resources

I plan to provide notes and PPT slides on the material I cover accessible from the course home page. These should suffice for the course. They will not be posted on Sakai, which will be used however for submitting programs.

Course Description
This course is intended for people who have learned to program. Its goal is to teach how to program well. The common programming strategy of beginners is to write the first solution they can think of without carefully identifying and weighing different alternatives. For all but the simplest problems, this approach of writing "quick and dirty" programs will take you to the debugging stage very quickly, but will make debugging slow. For large, complex programs, you need to identify multiple alternative solutions to the problem, choose an alternative that most directly solves the problem, and think carefully what your solution does, and how it works. The claim is that, although "quick and dirty" programming may produce a program faster, the concepts we teach will help you produce a correct program faster. Moreover, they will lead to programs that are easy to change and reuse.

We assume you have learned the following basic programming concepts: primitive types (integers, real numbers, Booleans), variables, constants, assignments, comments, expressions, arrays, loops, arrays, and procedures/functions/methods. These concepts are taught in most if not all introductory programming courses regardless of whether they teach conventional or object-oriented programming. This course will teach you the next-level programming concepts. These include objects, classes, interfaces, packages, inheritance, delegation, design patterns, exceptions, assertions, pointers, and formal correctness. These concepts will not help you solve new problems; rather, they will help you solve problems in new ways. The skills that will enable you to use these concepts will form a large part of the challenge you face in this course. After this course, you will have a much deeper understanding of the programming and learn some of the ideas that can make programming a science. We will be using Java as a vehicle for learning these concepts.

Target Audience

As mentioned in the course description, this course is intended for people who have learned to program. Its goal is to teach how to program well.

Prerequisites

As mentioned in the course description, we assume you have learned the following basic programming concepts: primitive types (integers, real numbers, Booleans), variables, constants, assignments, comments, expressions, arrays, loops, arrays, and procedures/functions/methods. These concepts are taught in most if not all introductory programming courses regardless of whether they teach conventional or object-oriented programming.

Goals and Key Learning Objectives
As mentioned in the course description, the goal is to teach how to program well. The common programming strategy of beginners is to write the first solution they can think of without carefully identifying and weighing different alternatives. For all but the simplest problems, this approach of writing “quick and dirty” programs will take you to the debugging stage very quickly, but will make debugging slow. For large, complex programs, you need to identify multiple alternative solutions to the problem, choose an alternative that most directly solves the problem, and think carefully what your solution does, and how it works. The claim is that, although “quick and dirty” programming may produce a program faster, the concepts we teach will help you produce a correct program faster. Moreover, they will lead to programs that are easy to change and reuse.

Course Requirements

The students must attend lectures and recitations, implement in-class recitation assignments, implement a semester-wide project, and take two midterm exams.

The material you learn makes little sense when you write small programs. This is why, in this class, you will exercise it in a large project you will build incrementally over the course of the semester. Each increment will be an assignment that builds on the software you have implemented as part of previous assignments. Because this is a programming course, unless all programs are submitted and work, you may not pass. Point values of assignments and programs will be in accordance with their length and difficulty.

As there is a cumulative project, there is no final exam in this course.

You must submit the source code of your program (with pledge signed) and screens showing executions of the program on test data. You may also do demos at certain stages of your project.

Examinations are closed book, notes and program listings; computers and collaboration are not allowed either.

Key Dates

Midterm 1: Tuesday Oct 15th, 2013 (in class)
Midterm 2: Tuesday Nov 26th, 2013 (in class)

Grading Criteria

Final grades will be based on the following formula:
22% Midterm Exam 1
22% Midterm Exam 2
10% Recitation Exercises
46% Project (Programming and Written Assignments)

The above percentages add up to 100%, but the instructors reserve the right to apply a 10% fudge factor to give consideration to things such as good class participation, stellar programs, and early submission and extra credit.

Course Policies

Students are required to attend each class unless there are extenuating circumstances. If such circumstances occur, you should access the class material posted for missed classes, and contact classmates to become aware of the announcements that were made.

Assignments are due at 11:59pm on each specified due date. Programs and homework assignments will be penalized 10% if one class session late and 25% if two class sessions late. No credit will be given for assignments submitted after the start of the second class period following the due date, although you are advised to submit work whenever it is finished as we can give your credit indirectly through the fudge factor. It’s always better to hand in a program, even if it is very late, than to hand in nothing, especially because later assignments will build on others. To help people who get irreversibly behind and cannot finish the last few assignments, we will allow you to shift assignment dates of the ones you finish based on the number of last few assignments you sacrifice. This means, you can apply the due date of the next assignment to the current assignment without late penalty, with the caveat that each time you do this, you sacrifice one more assignment at the end. For example, if there are 11 assignments, and you realize that there is no way you can finish assignment 4 a week after it is due, you can apply the due date of assignment 5 to assignment 4, and the due date of assignment 6 to assignment 5, and so on. But you have sacrificed the last assignment, assignment 11. Now let us say that you when you reach assignment 7, you realize you cannot finish it by the due date of assignment 8. You can shift dates one more time by sacrificing assignment 10. You are welcome to finish a sacrificed assignment and get it graded, but the grade you get will count as extra rather than normal credit. We expect each of you to finish each assignment in time – these rules are given because assignments build on each other.

As a large fraction (10%) of your grade depends on your performance in recitation exercises, you should attend all of them. You will be allowed to drop two of your recitation grades to take into account the fact that there may be extenuating circumstances for missing recitations.
Honor Code

1. You are encouraged and expected to discuss the assignments among yourselves.
2. You are permitted to discuss all aspects of the Java programming language with anyone.
3. You are permitted to discuss solutions at the design level but not at the code level. For example, you are allowed to tell others that you have separate classes in your program for scanning and evaluating expressions, or that you are using a loop instead of recursion for scanning, but are not allowed to show them Java interfaces, classes, while loops or other Java code in your solution. A general rule of thumb is that if you are communicating using a natural language, you are discussing at the design level, but if you are communicating using pseudo or actual code, you are discussing at the code level.

For details on how the Honor Code applies to programs, consult the handout 'Honor Code Observation in Computer Science Courses.'

Pair programming in recitations: There is considerable evidence to show that students who program together learn faster, infecting each other with their knowledge. Therefore, try the following: In each recitation, you do small exercises with a partner. Moreover, choose different partners in different recitations to ensure the “knowledge infection” proceeds rapidly – in each partner the person with whom you have not partnered the longest. In each pair programming exercise, one person should be coding and the other advising. You should switch coding and advising roles every 10 minutes. The adviser is free to look up information on the computer to help the coder.

Course Schedule

If possible, a schedule of topics covered by the course organized by course date or week number.
1. Course Information, Introduction, and Scanning
2. Object-first Introduction to Programming
3. State and Properties
4. Interfaces
5. Graphics/Geometric Objects
6. Composite Objects and Shapes
7. Trees, DAGs and Graphs
8. Inheritance
9. Type Checking with Inheritance
10. Arrays and Collections
11. Model View Controller and User-Interface Toolkits
12. Assertions and Exceptions
13. Adapters, Delegation
14. Animation, Threads and Command Objects
15. Composite Design Pattern, Visitor Objects

Disclaimer

The professor reserves the right to make changes to the syllabus, including project due dates and test dates. These changes will be announced as early as possible.