UNIVERSITY OF CALIFORNIA
Department of Electrical Engineering
and Computer Sciences
Computer Science Division

CS61B **P. N. Hilfinger**
Fall 2013

CS 61B: Introduction to Programming, Part II
General Course Information*

Instructor: Paul N. Hilfinger, 787 Soda Hall, 642-8401, hilfinger@cs.berkeley.edu

Introduction

Welcome to CS 61B. The CS 61 series is an introduction to computer science, with particular emphasis on software and on machines from a programmer's point of view. CS 61A covered high-level approaches to problem-solving, providing you with a variety of ways to organize solutions to programming problems: as compositions of functions, collections of objects, or sets of rules. In CS 61B, we move to a somewhat more detailed (and to some extent, more basic) level of programming. As in 61A, the *correctness* of a program is important. In CS 61B, we're concerned also with *engineering.* An engineer, it is said, is someone who can do for a dime what any fool can do for a dollar. Much of 61B will be concerned with the tradeoffs in time and memory for a variety of methods for structuring data. We'll also be concerned with the engineering knowledge and skills needed to build and maintain moderately large programs.

Discussion and Lab Sections

The TAs for this course will handle discussion sections and labs. We will maintain contact information about them on the web page. There will also be volunteer lab assistants staffing the lab sections (at least, we sure hope so).

I really don't care what discussion and lab section you choose to be in, but please make sure you do have them and that their TAs know about you. If you want to change sections, just clear it with the TAs involved.

*With contributions by Brian Harvey, Mike Clancy, Katherine Yelick, Jonathan Shewchuk, and John Canny.

Online Resources

The course home page will provide one-stop shopping for course information. All the handouts, homeworks, labs, FAQs, staff contact information, etc., will be posted there. The home page is `http://www-inst.eecs.berkeley.edu/ cs61b/fa13`.

Please *do not* print out online information that we hand out in class or that is available through the indicated copy stores. It is a waste of paper and ties up the lab printers.

The course newsgroup is Piazza. For most questions about the course, the newsgroup is the right place to ask them. The course staff read it regularly, so you will get a quick answer. That way, other students benefit by seeing the question and the answer. Don't forget to check the newsgroup before asking your question, just in case someone else has posted it.

If you are stumped by a particular error in a program you are writing for the class and want to ask us about it, please use the `bug-submit` program described in the Announcements section of the class web page. This will send us *all* of your code, as well as your question. Please do *not* use the newsgroup for this purpose. We will not feel obliged to answer debugging questions on this forum; it is almost always a waste of time, due to the lack of sufficient information.

The e-mail address `cs61b@cs` will send a message to the TAs and me. You can use it for correspondence that you don't want to send to the newsgroup. We all read it, so you will usually get a quick reply. If you send a question that is of general interest, we may post the response on the newsgroup (we will keep personal information out of it, of course). To talk with us, the best way is to come during regular office hours (posted on our doors as well as in the home page). Many of us are available at other times by appointment. Please don't be shy; web pages, e-mail, and news are useful, but it's still much easier to understand something when you can talk about it face-to-face.

When logged into our instructional systems for CS61B work, please make sure that you are using the standard configuration for the class—that is, the files `.bashrc`, `.emacs`, etc., that should have been in your accounts initially. If you must modify these, we suggest that you continue to have them read our scripts from the `~cs61b/adm` directory, so that we can easily propagate corrections to you. In any case, if you do modify these files, you are on your own.

Background Knowledge

Some of you may have thought that the stuff you learned in CS 61A/61AS was mere esoteric fluff inexplicably thrown at you to weed out the faint of heart. Not true. In fact, although the syntaxes of Java, Python, and Scheme are enormously different, the underlying computational models are surprisingly similar. You will find that almost all the "big ideas" you see in Java had their analogues in what you learned in CS 61A (indeed, one self-test of your understanding of the course material in CS 61B is to check that you see all the similarities). This course will assume you are familiar with the material in CS 61A, and there will be some references to the online 61A textbook (adapted for Python from Abelson, Sussman, and Sussman). If you haven't taken 61A, you may be confused sometimes, and you should make sure you review

the material (which you'll be able to find online in the CS61A web pages) early on in the semester.

All the instructional machines for this course will be running various flavors of the Unix operating system (generally, Ubuntu,) and it's a Really Good Idea for you to become familiar with it. There will be online information available from the class home page. Another good introduction is "A Practical Guide to The Unix System (3rd edition)" by Mark Sobell (Addison-Wesley, 1994) available at bookstores on- and off-line. Over the course of the semester, there are help sessions on various useful computer-related topics; we will put appropriate links on the course home page.

Is this the right course?

This is a course about data structures and programming methods. It happens to also teach Java, since it is hard to teach programming without a language. However, it is not intended as an exhaustive course on Java, the World-Wide Web, creating applets, user interfaces, graphics, or any of that fun stuff. Some of you may have already had a data structures course, and simply want to learn Java or C++. For you, a much better choice would be self-study, or (for C++) CS 9F "C++ for programmers," a one-unit self-paced course that will teach you more of what you want to know in less time. There is no enrollment limit for that course, and you work through it at your own pace after the first and only lecture. There is a similar course for Java (CS 9G). Finally, the 1-unit self-paced course CS 47B is for students "with sufficient partial credit in 61B," allowing them (with instructor's permission) to complete the 61B course requirement without taking the full course.

Wait-listed?

We would prefer to enroll all who meet the prerequisites. If you are wait-listed and do meet the prerequisites, stick around; people generally drop within the first few weeks, and you can get in. Those of you who *are* in and who drop: *please* inform TeleBEARS as soon as possible. If you didn't get in because of a lab- or discussion-section conflict, you will have to work out some way of attending one of the sections we'll be running. Do not worry about telling TeleBEARS; we'll try to work out what to do to get you in.

Course Materials

The only regular textbook in this class is *Head First Java, 2nd Edition* by Sierra and Bates (O'Reilly, 2005). This covers only the first part of the course. I'm not assigning a data-structures textbook for the other part of the course; they cost too much. Instead, I've provided all other material as readers. These are available online from the class webpage. Depending on demand, I may also make hard copies available from Vick Copy (1879 Euclid at Hearst); I'll announce this if it happens. Do *not* go to Copy Central for readers; I'm not using them this semester.

The Java text is not a reference book (it says so on page xxii). If you intend to become a serious programmer, you will probably want a full reference at some point. The official description of the Java core language is available online in The Java® Language Specification (Java SE 7 Edition) by James Gosling, Bill Joy, Guy Steele, Gilad Bracha, and Alex Buckley. This book does not contain a reference for the library, and the paper library references I've seen are less than satisfactory (and quite expensive). I'm inclined for now to stick with the online documentation for the Java library (there's a link to it on the class home page).

Laboratory and Discussion Sections

There will be one lab section and one discussion section per week. We have scheduled lab sections between the Wednesday and Friday lectures, and discussion sections between the Monday and Wednesday lectures. You should expect that we will present new material during the scheduled labs. Actually, you can do the projects and lab exercises any time, but we can only guarantee organized assistance during the scheduled lab sections in Soda. You should plan on attending the discussion sections. Tests will be returned in section.

Labs will be online, so you will be able to do much of the work ahead of time. One major purpose of the labs is to give your TA a chance to check up on you and to find out what people are and are not understanding. We've found that with the increasing ability to work anywhere has come an increasing tendency for students to go off by themselves and fall behind. Don't make this mistake. Keep up with homework and lab work and above all *let us know when you don't understand something!*

Homeworks are also online. We will not hand out assignments on paper; you are responsible for checking what's due when on the class homework webpage. Homeworks are individual efforts.

We evaluate projects and homeworks *on the instructional machines.* Very often, you will find that for various reasons (different software versions, different operating systems), a Java program will work on your home machine, but not on ours (the "write once, run anywhere" hype is just that: hype). So be sure to test your software on the instructional machines before submission.

Software

The official text editor and programming environment for this course is Emacs, which is available on the instructional machines, on GNU/Linux, MacOS X, and Windows. While the Emacs user interface may not *look* as nice, it works quite well, thank you, and provides all the editting, compiling, debugging, and version-control support you'll need.

I used to use Eclipse, a programming environment that's widely used in industry, but found its complexity and poor engineering made it a poor example for impressionable youth. In addition, it runs spectacularly badly when used remotely (i.e., run on a remote server from your home machine), whereas Emacs performs quite acceptably in this mode (so if you don't want to hassle with installing software at home, you can ssh into the instructional machines from home and use them easily). You are, however, free to use Eclipse or other

programming environments (JBuilder, Netbeans, Intellij IDEA, etc.) at home. Whatever you use, however, your submitted solutions must conform to our expected layouts, as indicated in the assignments.

This semester, we will be using Java (J2SE 7), which you can download for (or is provided with) Windows, GNU/Linux, and MacOS X. To obtain a version of the system for home use, go to the class web page, where you will find a link to Sun's download page. Under the circumstances, you may find it more convenient to simply use the instructional machines remotely by logging in from home. We should have information on this posted as well. The modified version of Sun's debugger that we use outside of Eclipse—called `gjdb`—is only likely to work on Sun's JDK; in particular, don't expect it to work at all on a non-Sun Java implementation. In particular, if you are using GNU/Linux distributions (such as Ubuntu), you may find that the GNU java runtime is the default, and that you will have to install the Sun version. You will need to download some `.class` files from us. Details will appear on the class home page.

We'll be using the version-control system Subversion for keeping your work. Version-control systems allow you maintain a series of "snapshots" of your files at various points in their development. Used properly, this provides you some back-up protection, so that you can recover previous states of your work when something goes wrong. We'll also use it for handing in work. In later, team-oriented courses (as well as the real world), version-control systems will help manage collaborative work. There are Subversion installations available for Windows, GNU/Linux, and MacOS X. Emacs provides a pretty simple interface that should work well for our purposes. Eclipse offers two rather persnickety interfaces: Subversive and Subclipse.

Unfortunately, experience shows that Subversion often shoots CS61B students in the feet, for some reason. Therefore, this semester, we'll be trying out a simplified front-end program for Subversion called `hw`. You can use it at home, too, if you first install Python 3 (at least on Linux or Mac). You will still need to install Subversion; you just won't use it directly.

Computer Accounts

The CS 61B scheduled labs will all be held in 275 Soda, which contains Intel workstations running Solaris. At other times, you can use any computer that is not being used for another scheduled lab. You will receive a computer account form in lab on Tuesday. Extra forms will be available in 385 Soda after that. Information on computer facilities and computing from home should be available via the class Web page.

You must electronically register the account you intend to use for handing in assignments (only one account, please) during the second week (by Friday's lecture). The class web page contains a link with which you can register your account and perform other adminstrative actions. Please use it to give your registration information and to generate the keys you need to work remotely and hand in homework assignments.

Homework and Programming Assignments

There will be many lab activities that record some of your work, more ordinary homework for you to do outside the lab, some of which includes small programming problems, and four larger-scale programming projects. You will turn in everything electronically. Be sure you have an account (registered for CS61B) for that purpose.

Testing and Grading

In addition to homework, there will be at least two tests during the term, and a final. All tests are open book, open notes, and may cover any material whatever (however, to prevent out-and-out mutiny, I am generally reasonable about the material selected).

The programming projects will count for a total of 100 points, and written homeworks and labs for 20 points. The final will be worth 46 points and other tests will count for a total of 34 points. Your letter grade will be determined by total points out of the possible 200:

185	170	160	150	140	130	120	110	100	90	80	75	< 75
A+	A	A-	B+	B	B-	C+	C	C-	D+	D	D-	F

In other words, *there is no curve;* your grade will depend only on how well you do, and not on how well everyone else does. For your information, University guidelines suggest that the average GPA for a lower-division required course be in the range 2.5–2.9, with 2.7 (B-) being typical. This corresponds to getting 50% on tests (typical for my courses), 75% on projects, and 100% on homeworks and labs (on which you get full credit simply for turning something in that displays reasonable effort on all questions).

If you believe we have misgraded an exam, return it to your TA with a note explaining your complaint. We will regrade the entire test. You should check the online solutions first to make sure that this regrade will make your total score go up.

I will grant grades of Incomplete *only* for dire medical or personal emergencies that cause you to miss the final, and only if your work up to that point has been satisfactory. Do *not* try to get an incomplete simply as a way to have more time to study or do a project; that is contrary to University policy.

Inevitably, some of you will have conflicts with the scheduled exam times. I will arrange for alternative test times for those people who have sufficient cause. Sufficient cause includes exams scheduled at an overlapping time in another course, medical or family emergencies, or important religious holidays (however, I believe I have avoided all of those). Popular reasons that are *not* sufficient cause include having job interviews, having a plane ticket that you (or your parents) bought without consulting the schedule, having exams or assignments in other courses at nearby times, being behind in your reading, being tired, or being hung over. With the obvious exception of emergencies, you must arrange alternative exam times with me well in advance.

Etiquette

Technology is wonderful, isn't it? Why, now you can read your mail, surf the net, order new books and clothes, and even do your homework anytime and anywhere. It's great, but don't do it in my class. In fact, I'd suggest not doing it in any of your other classes, either, nor in any public lectures you might attend.

The problem is that while all this connectedness is nice (for you), it is not particularly *discreet*. If a student is doodling in his old-fashioned notebook (the kind that uses paper), then at least the speaker and the most of the audience won't notice. But you've probably noticed that you can hear someone typing from anywhere in the room. And the moving images on screens are much more distracting to those sitting behind than are movements of a pencil[1]. In short, the use of laptops (and cell phones) in class *advertises* disrespect of the speaker. As a service to you and the public at large, I have determined to help you break the habit of doing so.

In passing, I should mention that there are old-fashioned ways of showing disrespect as well, which you should also avoid. Reading the newspaper during class is one example (especially when you hold it up in front of your face so that everyone can see what you're doing). Gossiping with your neighbor is another (especially since nobody seems to know how to whisper anymore—possibly as a result of early earbud-induced hearing loss).

Policy on Collaboration and Cheating

I strongly encourage you to help each other on homework assignments and in labs. Ordinary homework and labs are not seriously graded: you get points for showing evidence of substantial effort. Naturally, though, it is in your best interest *not* to take advantage of this fact, and to treat homework seriously.

The four projects are *individual* efforts in this class (no partnerships). Feel free to discuss projects or pieces of them *before* doing the work. But you must complete and write up each project yourself. That is, feel free to discuss projects with each other, but be aware that we expect your work to be *substantially* different from that of all your classmates (in this or any other semester).

Copying and presenting another person's project or test work as your own constitutes cheating. Electronically submitted programs are particularly easy to check for copying or trivial changes, and we will be doing that. I will report any incident of cheating to my departmental chairman and to the Office of Student Conduct.

I realize that CS and EE programs at Berkeley are very intense, and that students are often under extraordinary pressure to make deadlines. But deadlines are a fact of life, and will persist after college. The trick is to get ahead of them. You can seek advice from the staff early if you feel yourself getting behind in something. Knowing where and how to get advice on things you don't understand is a skill everyone needs to succeed in the real world. Finally, those of you who are EECS freshmen and sophomores should realize that even a poor

[1] What's that? You're actually using your laptop to take *notes*, you say? Yeah, right.

grade in one introductory class will have little effect on your final GPA; for you, there's even less to gain from yielding to temptation.

Lateness

We will give no credit for written homework turned in after the deadline. Please do not beg and plead for exceptions; an individual assignment is worth too few points to justify your groveling at the my feet (a comment that probably applies to individual test questions, as well). You can miss an assignment or two and still get your A+.

On programming projects, we are a tad more lenient. I'll penalize an assignment 5/12 percent for each hour it is late, rounded off in some unspecified fashion. I'll give you some grace hours for free. They work like this: on the first project, you get 24 hours of additional time past the deadline before we start counting your project as late. On the second project, you also get up to 24 hours *plus* any leftover late time you had from the first project. Likewise, on the third project you get 24 hours, plus any remaining late time you didn't use for the second project. This arrangement is an experiment. I used to just give a total of 72 free late hours for all four projects, but found that people tended to blow all their time on the first project and, in fact, to delay working on it in the expectation of using late hours.