Syllabus and Policies
EECS 381: Object-Oriented and Advanced Programming
Winter Term, 2010

Professor

David Kieras, kieras at the usual umich address, 3641 CSE Bldg., 763-6739

Required Textbooks

Kernighan, B. & Ritchie, D. *The C Programming Language*. (2nd edition), Prentice-Hall, 1988. (good used, but 2nd edition only!)
Stroustrup, B. *The C++ Programming Language*. (3rd or Special edition). Addison-Wesley, 1997.

Recommended References

Harbison, S., & Steele, G. *C: A Reference Manual*. (5th edition), Prentice-Hall, 2002. (3rd or 4th edition is still useful)
Josuttis, N. *The C++ Standard Library: A Tutorial and Reference*. Addison-Wesley, 1999.

Course Communication

This course will use computer-based communication heavily. You need to be adept with using email, web browsers, and downloading/uploading files. You must have access to facilities that will enable you do these activities easily and conveniently. Depending on the platforms, software, and techniques you use for up/down loading of files, you may have text file conversion problems, and you will have to know or learn how to deal with them.

Course online information and files. There is a course home page which can be reached with a web browser from the EECS Department courses home page, or the CAEN course listings: http://www.engin.umich.edu/class/eecs381/. There is also a file server, at afs/engin.umich.edu/class/perm/eecs381. All project documents, lecture notes, handouts, etc., will be posted in these two places, as well as all kinds of goodies, making it your first stop for help or getting answers. Set a bookmark for these sites, and plan on checking the web page frequently. If you ask about material that is already on the web pages, you should expect either no answer or a "go read" reply - I have to save my time for the *new* and *difficult* questions.

Course email mailboxes. I will send official announcements to the UM email group **eecs381announcementsW10** and all students are supposed to get them in their email. All students *must* join this email group immediately; *it is a course requirement*. All official course announcements will be sent to this group and you are required to read them. Only the professor can send to this group. You can join it in several ways, but the most convenient is the web page http://directory.umich.edu, or consult other documentation. You can use the same facility to resign from the list if you are no longer in the course.

If you need help with technical questions about the projects, lectures, or the coursework, or administrative problems, send a message to eecs381help@umich.edu. It will be forwarded to the professor (and TA, if any). This is the quickest way to get help, since my email filters will call it to my attention promptly. Setting a shortcut to this address is a good idea; you may be using it a lot. Don't send a message directly to the Professor's email address unless you are willing to risk being last-priority for a response. It can be inconvenient, but if I send you a follow-up question, please respond to eecs381help instead of directly. When the email traffic is heavy, this makes it much easier to keep track of things.

About the Course

General approach. To make sure we have time to help you learn and enjoy some neat programming concepts, the policies and mechanics of the course are quite rigid; they are spelled out in glorious and awful detail in this tedious document. You are required to be aware of these rules and to follow them. This way, instead of fussing over how things will be done, our time can be spent on substantive, useful, and fun work.

Course goals. This elective course introduces advanced concepts and techniques in practical C/C++ programming. We will start with a quick, but deep, introduction to important topics in C programming, and then the course will emphasize object-oriented programming with the use of single and multiple inheritance and polymorphism, and using the Standard Library algorithms and containers. Key ideas in object-oriented analysis and design and common design patterns will be introduced. Programming projects will focus on learning and using techniques that are valuable for professional practice in developing and extending large scale or high-performance software relatively easily. In addition to these content goals, an important function of the course is to help students develop good programming practices, style, and skill, with as much personalized coaching and critique of individual student's code as possible. In short, the course is intended for those who want to become outstanding programmers.

Course topics. The course will cover at least the following topics. See the separate document for the exact schedule for the course.

1. Short course on programming concepts in the C language:

- The separate compilation model, declarations vs definitions, function prototypes. Internal/external linkage and the linker, the one-definition rule, undefined behavior.
- Incomplete declarations, pointers, function pointers, void pointers, type safety, type casts.
- Memory allocation and management, basic C stream and file I/O.

2. C++ fundamentals:

- C++ as a "better C".
- Exceptions and exception safety, namespaces, classes, static members, classes that manage resources.
- Operator overloading, generic programming, basic use of function and class templates.
- The C++ Standard Library: Organization, containers, algorithms, iterators, function objects, binders, adapters, strings, streams.
- Inheritance, substitution principle, multiple inheritance, polymorphism, virtual functions.
- Run-time type identification, dynamic casts, up-, down-, and cross-casting, RAII, "smart pointers"

3. Object-oriented programming concepts:

- Basic class design principles – collaborations and responsibilities; separating interface and implementation; decoupling.
- Object-oriented principles and techniques – using a polymorphic class hierarchy; abstract base classes for common interface.
- Major object-oriented idioms and design patterns – providing extensibility and code stability simultaneously.

Prerequisite and workload expectations. A C or better in EECS 281 is the required prerequisite for this course. Since a grade of C or better is the minimum grade for required courses in the CS and CE degree programs, if you received a grade of **C-** (C-minus) or lower in 281, you do not belong in this course, but should be retaking 281 (or any other course with a C- or lower grade). Since the course concerns the concepts and techniques for building and maintaining complex software, you will be expected to design, write, test, and debug a relatively large amount of code throughout the semester, making this a very high workload course. Experience suggests that if you got only Cs in 280 or 281, you are at great risk and will have to work especially hard right from the beginning to keep up, and be prepared to drop the course early in case you are not keeping up. Even many students who earned good grades in the prerequisites have been surprised by the workload and have either been forced to drop the course or accept a failing grade (C-, D, or E). In most cases, these problems began with procrastination disasters on the first project - many students apparently think they are better programmers than they really are, and put off the first project until it is fatal. To succeed in this course, you will have to give it extremely high priority and lots of time right from the beginning. See the "How to do Well on the Projects" document.

Course format and attendance. There are two lectures per week, one discussion session per week. The lecture topics and reading assignments will be announced in the schedule posted and revised on the course web site. The purpose of the lectures and the discussion sessions is to explain the material and to answer your questions. When there is a lot of material to be presented, the discussion section time will be used as "overflow" lecture time, so you should plan on attending - it is part of the course. You are expected to have studied the assigned textbook material yourself prior to the lectures on each subject. There will be several programming projects (one due approximately every two weeks), a midterm exam, and a final exam; their due dates are given in the course schedule which will be maintained on the course web page. Changes to the schedule will be announced in lecture and via the announcements email group, and updated on the course web page.

You should plan on attending the lectures and discussion sections always. You may choose to skip class, but you are still responsible for all material and announcements presented in the lectures and in the discussion sections. Material not in the textbooks will often be presented in lecture or discussion; you are responsible for this material in addition to that in the assigned readings. Your projects will be evaluated in terms of whether they take advantage of the course material properly, not just whether they happen to work correctly or are written well by other standards. So skip the lectures and discussion at your own risk. Experience shows that students who blow off the lectures and discussion usually have a tough time with the projects and the exams and lose lots of points on code quality evaluations; a *failing* grade is a common result.

When feasible and useful, the course web pages will include materials such as handouts, lecture outlines, and code examples. In general, this material will not be comprehensible without having attending the lecture; it will be provided to supplement, not replace, the lecture presentation.

Programming Environment. C has long been standardized, and the standard was revised in 1999 (so-called "C99"), but at this time many C compilers lack major parts of the C99 standard. For C programming, we will be assuming the long-used "C89" standard which is the most familiar C dialect, corresponding closely to the Kernighan & Ritchie book we will be using. C++ was standardized

in 1998 by the ANSI and ISO standards organizations (a revision is underway). The major vendors' current C++ compilers and libraries are very close to meeting the C++ Standard. Unfortunately, there are small differences remaining between different compilers and libraries in exactly what parts of the Standard they do and do not meet. These differences will be dealt with in this course as needed, but the following rules apply: *This course will teach Standard C and C++. Except when specifically stated, you must use only Standard C and C++ in your projects. You may not use any C and C++ facilities that are not part of the Standard.* In case of doubt, look it up or ask.

You may use any C/C++ programming environment for this course, as long as you can conform to the above rules about the Standard. Recommended are the most recent versions of gcc (4.x) under Unix or Linux, Apple's Xcode Integrated Development Environment (IDE) for Mac OS X which currently uses gcc 4, and for Windows, the ".Net" versions of MicroSoft Visual Studio C++ (v. 7 or later). The project grading system runs under Unix, using a version of gcc 4 and options which I will announce. You will have to use Unix to submit your projects; instructions will be provided, but you should have the basic Unix skills involved.

Debugger usage is required. In all of the programming environments, I expect you to learn and make use of a debugger right from the beginning. The value of this tool for serious programming cannot be overstated. It is easier to learn a debugger on simple projects than on the complex projects where it becomes extremely valuable, so start using it right away. The modern IDEs such as Xcode or Visual Studio have amazing, usable, and powerful graphical debuggers, and by themselves are good reasons to use an IDE instead of the clunky traditional Unix command-line tools.

Programming Projects

How to do well on the projects. There is a separate page on the course web site on this topic. You should study it carefully when you start on the first project; you are responsible for knowing and acting on the content of that document. Here will be stated the basic rules and regulations for the projects, but the *How to do well* page contains further details on the course policies as well as advice.

What the projects require. The project assignments will specify not just the behavior of the program, but also many features of the design of the program and the techniques to be used in it. This is to ensure that you have an opportunity to learn program design concepts and the many specific techniques that will be presented in this course. Getting correct output is not enough - the correct output must be obtained in the specified way and using the concepts presented in this course. The project assignment specifications will be very detailed. You are expected to read them carefully and ensure that your work meets the specifications. You are expected to ask questions if the specifications are not clear. Corrections & Clarifications to the specifications about the projects will be posted on the course web pages and frequently updated while the projects are being worked on. These Corrections and Clarifications are officially part of the project specifications. Check these often during the project work period to make sure you haven't missed something. Finally, note that throughout, you are expected to design and write high-quality code that makes full and appropriate use of the course material, even when the specifications do not specifically require it.

Project grading. Projects will be graded in two ways: Using an *autograder* I will test your project for correct input-output behavior and correct component definition and behavior. The autograder is extremely strict and demanding. The goal of the autograding will be to give you full credit if your project program (a) compiles correctly; (b) executes without error and produces *exactly* the specified output; and (c) meets the project specifications for the behavior and structure of the components. To get full credit, you must meet all three of these requirements.

Using *human grading* (my eyes and brain), I will evaluate your actual code for (a) quality of its writing (including much more than just "style"); (b) whether specified concepts and techniques were used; (c) whether course guidelines for quality code and design were followed; and (d) whether your design was well thought out and implemented. Because the human grading both is very laborious and generally provides the most useful feedback for becoming an advanced programmer, it will be weighted much more heavily than the autograding scoring.

Six projects are planned. The first five will involve autograder grading. The sixth will involve only human grading. The first and third will involve both human and autograder evaluation, and the following rule will apply: The autograder score will count 30% of the project score; the human grading score 70%. However, to qualify for human grading, your project autograder score must equal or exceed a threshold. The threshold value will be the smallest integer value that is at least 80% of the maximum autograder points (bonus points are not counted towards the threshold). For example, if the autograder awards a maximum of 21 points, then the threshold would be 17 points. If your project does not qualify for human grading, it will be awarded 0 points for the human grading portion of the score.

You are expected to write code that follows the quality and design principles in this course even if the projects will not be given a full human-grading. Therefore, the projects planned for autograder-only scoring (the second, fourth, and fifth) may be subject to a special *spot-check* evaluation. At my option, I will check very briefly to see if required design tasks and components appear to be actually

attempted, and whether especially serious code quality problems are present. If problems appear in this spot-check, your autograder score will be reduced by a minimum of 10% of the maximum points per problem, and perhaps much more; having the score reduced to zero is a possibility.

Project due dates. The due dates and times for the programming projects are listed on the course schedule on the web page and on each project assignment document. In case the due date must be changed, the date on the project document takes precedence over the date listed in the course schedule, and announcements of due date changes (either in lecture or by the announcements mail group) take precedence over the date on the document.

The autograder used for this course allows you to submit projects as often as you like, but you only get feedback on the scores for the first two submissions per day. Each submission replaces the previous one. The last submission on file is your official submission, and the date and time of this last submission is the official date and time of your submitted project. See the web page "Autograder Information and Policies" for more details.

Projects will not be accepted after the due date - there will be no late submissions, no "late days", and no "free late days." *The project must be submitted by the announced due date and time or it will not be accepted*. Take this seriously - you really will get zero points if your project is submitted even a second after the deadline. You will have to start the projects right away to be sure of getting them in on time, and you should do everything in your power to avoid working right up to the deadline - under such conditions, it is too easy to make a mistake that results in a zero score for your final submission, and I won't allow you to fix it.

To encourage you to start the projects early, there will be a bonus for getting the project in ahead of the due date. Additional points in the amount of 10% of your autograder score will be awarded if your final submission is two or more full days before the due date, and 5% of your autograder score if your final submission is one full day before the due date. Except for the last project, bonuses apply only to the autograder portion of the project score.

Turning in your projects. Because there are many students and projects to keep track of, I must insist that you submit your projects according to my instructions. I will not be responsible for any projects not turned in properly, and I will reject any claim of a missing project or grade if you have not followed the stated procedures. *In case there is some problem that is not your fault, you should always keep a complete copy of everything that you turned in for a project and all messages and grading results about your project*. Instructions for submitting your projects will be provided on the course web pages and with the individual projects.

Extensions to project deadlines. Experience shows that the only fair policy on extensions to the project due dates is a strict one that is enforced uniformly. Here are the rules:

- Only the Professor can grant an extension.
- Only documented medical problems or personal emergencies will be considered as a reason for an extension. If this applies, please be prepared to provide some documentation.
- You are expected to contact the Professor before the assignment due date if it is at all possible.
- Substantial amounts of late work resulting from long-term problems will be accepted only if it is completed on a schedule that you and the Professor agree to and document in email messages. You must contact the Professor to arrange this as soon as possible.

Extensions will definitely *not* be granted for reasons such as the following:

- Equipment breakdown, network congestion, difficulties in getting access - these are part of computing life, and should be planned for - don't work up against the deadline, because then there is no room for the unforeseen – which always happens.
- You accidentally erased your files, lost your disk , etc. You should have backed them up! Computer disk drives are just reliable enough to trick you into thinking you do not have to back them up. Form good habits of keeping a second copy of all of your work, and saving it at least at the end of every work session.
- Conflicts with job interviews, athletic events, vacations, other course work, or other scheduled events or activities - you can complete the assignments in advance (and get the early submission bonus as well!). In the event of unscheduled, unplanned events which you believe are required of you, check with the Professor in advance, as soon as you know of the conflict.

Reading Assignments

Brief papers required. We are using books containing considerable practical wisdom, which you will definitely benefit from reading. Also, I want the class time concerning the textbooks to be spent on discussion, clarification, elaboration, and explanation of the textbook material. I do not want to recite the book for you; your reading has a much higher information transfer rate for the basic concepts than your listening to me recite.

But for this approach to work, it is vital that the students read the material before it is discussed in class. An effective way to do this is to require that students demonstrate that they have read the material by writing a *very brief* paper about each reading assignment. The assignments will be announced in class or appear in a schedule on the web site; each assignment has a due date on which you are to hand in at class a paper on the reading assignment.

Paper content. The content of the paper must meet the following criterion: *It must be obvious to me, upon skimming through your paper, that you read all of the assigned reading*. There are no other requirements for the content of the paper. You can simply write an outline, list questions you have about the material, or you can relate the material to your own interests or experiences. It just has to be obvious that you read the entire assignment.

Paper length. The paper should be at least one page long, but should not be any longer than both sides of one piece of paper if handwritten, or no more than 2 pages if typed or printed. Printed output on both sides of one sheet of paper, 10 or 12 pt Times, margins no more than 1", single or double-spaced, is preferred. My experience has been that if you write less than a true full page, such as scribbling a phrase on every other line of one side, your paper will probably not pass the criterion. So, the paper should be at least one real page in length. If you hand in more than one piece of paper, *you must staple them together*. Electronic submission is normally not allowed; if it seems required for a special situation, you must get my permission, and it applies to that one assignment only.

Number of assigned items. There is only one paper due on a class period, even if the reading assignment includes multiple items. For example, if the assignment is to read chapters 2, 3, & 4, and a downloaded handout, this equals *one* reading assignment, and only one paper of 2 pages is required. However, you must be sure that your paper covers *all* items in the assignment.

Grading of reading assignment papers. Since these papers are important, I will be including them in the course grading. So that you can set your priorities, a *satisfactory* paper turned in on time (on the due date) will be worth 1 point, a paper turned in late will be worth 0 points, and a paper not turned in at all will be worth -1 points. A late paper will be accepted only if turned in no later than one week from the due date. In the grading, the total set of papers will count 8% of the total course grade; thus the reward for keeping up on the readings is substantial. Skipping reading papers can cost you a good grade, because a missing paper cancels out an on-time paper, resulting in zero points for the pair.

Each paper will be marked satisfactory or unsatisfactory, and returned to you in class. Unsatisfactory papers are those that do not meet the "obviousness" criterion, and will be counted as not having been turned in at all. They must be redone, and resubmitted *within a week* after they were available to be returned, whereupon they will be scored as late.

Grades and Grading

Exams. There will be two exams on the date and at the time shown in the course schedule. The first exam time is awaiting room scheduling, and will be announced as soon as possible. You are expected to schedule other events such as trips or job interviews to avoid conflicts with the exam dates. The only acceptable excuses for missing an exam on the scheduled date and time are documented medical problems or personal emergencies. An unexcused absence at an exam will result in a grade of zero on the exam.

Questions and disputes about grading. I develop a grading policy for each exam question, project, and the course as a whole, and attempt to apply it consistently. But grading mistakes happen: A grading mistake is when I fail to follow our own grading policy or specifications correctly. If I misgrade your work or make a clerical error, I will fix the problem with a smile!

But changes to grading policy will be very rare. In general, I will only alter a grading policy if I decide I should alter it for everyone (not just you) and regrading everybody's work accordingly would make a practical difference in the course grading.

Timely notice required. I will not correct any grading errors that are not brought to my attention in a timely manner. Any grading errors must be brought to my attention within a week of when the results were made available (e.g. within a week of when exams were handed back in class to anybody who was there).

Course grade determination. The course programming projects make up 60% of your course grade; six projects are planned at 10% each. Any modifications to the number and weight of projects will be announced. The course exams constitute 30% of your final course grade, 15% for the midterm exam and 15% for the final exam. The pretest counts 2%, and the reading assignment papers constitute the remaining 8% of your grade.

This course will use a noncompetitive grading scheme that awards grades in a way that does not limit the number of good grades (e.g. A's) that will be awarded, and does not necessarily result in any bad grades (C's, D's, or E's). The concept is that your grade is based on the extent to which you have mastered the course material, where the criterion for mastery is based on the best scores produced by the class. A separate criterion will be determined for each of the projects and exams.

Specifically: On each exam and project, the average of the top 10% of the scores in the class will be the *criterion* score. Each of your raw scores on an exam or project will be converted to a percentage of the criterion by dividing your raw score by the criterion for that item. A negative total for papers will be converted to zero. Your final grade will be determined by calculating your average percent of criterion, weighted as described above for reading papers, projects, and exams. Your average percent of criterion will be converted into the letter grade depending on highest of these ranges that it fits into:

A range: >= 90%; B range: >= 80%; C range: >= 65%; D range: >= 50%; E range: <50%.

The lower third of each range will be awarded a "-"; the upper third a "+." Thus to get program credit for this course, your average must be at least 70%, which is the minimum for a grade of C; an average below 70% is a C- or lower, which is not acceptable for program credit.

This scheme does not arbitrarily limit how many A's (or any other grade) will be assigned. However, you still have to work hard to get an A, because the criterion is set by the students who do very well. The criterion score automatically adjusts for unexpectedly easy or difficult assignments or exam questions. I will announce criterion scores for each project and exam, so that you can calculate how you are doing.

Subjective impressions of your *performance* in the course *may possibly* be taken into account if you are *extremely near* a grade cutoff, but at most only to the extent of pushing you over the cutoff into the next higher grade category. Being close to a cutoff does not mean you will get a higher grade - my decisions on such borderline cases are final and not a subject of negotiation. Note that if you are that close, a little bit more effort on your part would have won the higher grade, because in no case will you be awarded a lower grade than your scores indicate.

Consistent with University grading policy, I will rarely give an Incomplete grade, and only because you have missed a small part of the course work for an excused reason. You will definitely not be given an Incomplete simply to avoid getting a bad grade.

A final word on grades: Your grade in this course is based only on your performance on the course work, following the grading system described in this document. If you need a certain grade, you will have to earn it by paying attention to the course throughout the semester, keeping up with the assignments, and doing the work well enough. Experience shows that students who shoot only for a passing grade often fail this course - it is really intended for people who want to learn these concepts well, not for those who just want to scrape by.

If you are dissatisfied with your grade, be informed that: *Your grade is not a subject of negotiation*. Our responding to individual appeals for leniency or special consideration in the final course grade would be inherently unfair, and simply will not be done, no matter what the situation. If a poor grade causes a serious problem for your standing or progress in your program, you should discuss the implications and effects with your academic program advisors, not the teacher of this course.

Academic Integrity Policies

Collaboration policy. Much real programming work is done in teams, but to be an effective team member, you yourself must possess the basic programming concepts and skills. Furthermore, you can learn these concepts and skills only if you undergo all of the experience on your own. Therefore: *No collaboration on the course projects is permitted unless specifically stated in the project assignment*. Thus the default is that you must do all of the project work by yourself. This means you must design, write, debug, and test your own code for each project. You may not in any way exchange any information with anybody else specifically about the code in your project or how you have designed and tested it. The course teaching staff (the Professor and TA, if any) are the *only* people who can look at your code, or tell you what should be in it, or help you test it. It is permitted to get certain kinds of help from other people, but if you do not understand what is permissible, *you are required to assume that all discussion with other people about the projects is forbidden until you clarify it with the Professor*.

Collaboration of any sort on reading assignment papers is **not** permitted. You must read the assignments yourself and write your own paper. You may certainly discuss the contents of the readings with anybody at any time.

Cheating on the examinations and projects will be taken very seriously. Advanced programming is a highly marketable skill and computing is central in our society and technology. *If you cheat in this course, you are making a deliberate decision to obtain a false credential for programming skill and thereby defraud, mislead, and possibly endanger your future employers, coworkers, and customers*. Programming is too important for dishonest programmers to be tolerated. If we detect or suspect cheating, fraud, or forgery on exams or on programming projects, we will make a formal report to the Engineering College Honor Council, which will then independently investigate and recommend action to the College Administration.

You are expected to do the project work honestly. Subverting, deceiving, or interfering with the grading system, regardless of the reason or situation constitutes cheating. Attempting in any way to get an undeserved grade through misrepresentation, forgery, or misuse of the computing facilities, also constitutes cheating.

You must design your code yourself. You are expected to do your own design work on all the programming projects. Discussing the specific design of your code with another student or somebody other than the course professor or GSIA is not allowed. Not only will you benefit more by thinking through the issues yourself, but in many cases collaboration between students on project design has resulted in all of them doing poor quality work - your fellow students in this course are generally not any better at it than you are! This prohibition also applies to tutors, regardless of their source or status. The only allowed sources for design discussions and ideas are the course teaching staff and the supplied materials for the current semester of this course.

You must write your code yourself. You are expected to do your own coding work on all the programming projects. Submitting somebody else's work as your own, with or without the other person's knowledge, constitutes cheating. You may not describe your code to anyone else in any way, or get descriptions of any sort from anyone else of what code to write. This prohibition applies to tutors, regardless of their source or status. The only allowed sources are the course teaching staff and the supplied materials for the current semester of this course.

You must keep your code private. Making your project code available to somebody else is enabling them to cheat; you are not permitted to make your code public in any way. If you use a source code management or version control server when collaboration is permitted, it must not be publicly accessible.

You must test your code yourself. The skill of testing a program for correct behavior is as much a part of programming as creating the code. Part of this skill is creating sets of test inputs or conditions that reveal bugs in the program. You may not share such test sets, nor may you compare your program's behavior with the behavior of someone else's program.

Helping each other with concepts and general issues is good. Sharing expertise and helping each other learn is not cheating and can make better programmers of all concerned. Note that the course grading is not competitive, so helping each other learn will benefit all of you. Some examples of persons A and B helping each other in acceptable ways include:

- A explains to B the concepts and techniques required or involved in the project, at the same level of generality as presented in the textbooks and lectures, without getting into the specifics of the program or code design.
- A helps B understand a specific bug or error message B is getting, without looking at B's code.
- A helps B with the specifics of C++ syntax or library function usage, without providing specific project code.
- A helps B with the use of a computer or of a programming environment, at a general level.
- A reminds B about the general concepts of testing programs, as presented in the course materials.

When you can use other code sources. Learning to find and use sources of information about programming is not cheating if done within certain restrictions. You can use general concepts, techniques, and advice from any source that you wish. But you can use *actual code* from sources other than your own mind only under the following rules:

- You may freely use code presented in ***this semester's offering*** of this course and its materials from lectures, discussion sections, the course texts, handouts, and web pages. ***Note***: You are still responsible for the correctness of your program; a error due to a bug in supplied code is your error, if you use the code.
- You may not use code from any other source. The forbidden sources include, but are not limited to, newsgroups, other people not associated with this course, other programs or projects, either not associated with this course or from other courses (either the previous semesters of this one, or other courses here or at other universities).
- In case of doubt, you are required to ask us for a decision about the appropriateness of using the code. *You must assume that if you did not write the code yourself, you are not permitted to use it*.