Syllabus and Policies
EECS 381: Object-Oriented and Advanced Programming
Fall Term, 2019

Professor

David Kieras, eecs381help at umich and edu, kieras at the usual umich address, 3641 BBB, 763-6739

GSI

Jacob Hage, eecs381help at umich and edu, jakehage at the usual umich address.

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*. (4th edition only!). Addison-Wesley, 2013. (This is the C++11 edition.)

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*. (2nd edition). Addison-Wesley, 2012. (Some C++11 coverage)

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: http://www.umich.edu/~eecs381/ which will be the primary information source for the course. There is also a file server, at afs/umich.edu/class/eecs381 linked to from the course web site. 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 pages frequently. If you ask about material that is already on the web pages, you should expect either no answer or a "go read" reply. *Good programmers **read***! So we will expect you to read first, then ask questions, not the other way around.

Announcements. I (the Professor) will announce important events such as when the autograder is open for a project, or changes to the course schedule. Due to student privacy policies, I will make official announcements using an appropriate facility. Watch for an email about this. If you are enrolled in the course, you should get these announcements automatically. All official course announcements will be delivered this way and you are required to read them. Except for announcements, all information will be provided via the course web site.

Email help. If you need help with technical questions about the projects, lectures, or the coursework, or administrative problems, send a message to eecs381help at umich and edu. This is the quickest way to get help, since both the prof and the GSI are in this group, and our email filters will call it to our attention promptly. Setting a shortcut to this address is a good idea; you may be using it a lot. Be sure to put a unique string descriptive of your question in the subject line so we don't see a dozen "EECS 381" subject lines in our in-boxes. Reserve directly emailing me or the GSI only for special situations, and if you do, be sure to put "381" in the subject line. It can be inconvenient, but if we send you a follow-up question, please respond to eecs381help instead of directly to us. When the email traffic is heavy, this makes it much easier for us to keep track of the questions and answers.

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; don't bother to ask for exceptions or special treatment. The idea is that instead of fussing over "administrivia" our time can be spent on substantive, useful, and fun work about advanced programming.

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 generic programming and 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 start becoming 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, copy and move semantics.
- Operator overloading, generic programming, basic use of function and class templates.
- The C++ Standard Library: Organization, containers, algorithms, iterators, function objects, binders, lambdas, 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.

4. Principles and practices for high-quality code:

- Using language features to provide single points of maintenance.
- Using function call hierarchies and classes for readable code organization and information-hiding.
- Using good code layout, ordering, and commenting to improve readability and maintainability.
- Using simple, straightforward code for clarity, ease of development and debugging.
- Avoiding egregious inefficiency, premature optimization, and premature "pessimization."

Prerequisite and workload expectations. A C or better in EECS 281 and 370 are the formal prerequisites for this course. 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 the workload extremely high or even “insanely" high. 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 you should 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 — lulled by the smaller projects in previous courses, 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 on the course website.

Course format and attendance. There are two 2-hr lecture times per week, no discussion section. The topics and reading assignments will be announced in the schedule posted and revised on the course web site. Actually the class time is a mixture of lecture and discussion — I will often present information that extends the readings, and most of the time will be spent answering your questions. You are expected to have studied the assigned textbook and handout 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 and updated on the course web page.

You should plan on always attending class. You may choose to skip class, but you are still responsible for all material and announcements presented in class. Material not in the textbooks will often be presented in class; 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, including that presented in class. So skip the class sessions at your own risk. Experience shows that students who blow off the class times, or spend them doing other work or playing on their laptops instead of listening and asking questions, usually have a tough time with the projects and the exams and lose heavily on code quality evaluations; a *failing* grade is a common result.

The course web pages include lecture notes (outlines), and code examples. In general, this material is provided to supplement, not replace, the class presentation. If some topic is not clear to you, looking at the more complete examples in the lecture notes and code examples may help.

Programming Environment. C has long been standardized, and the standard was revised in 1999 (so-called "C99"). For C programming, we will be assuming a mixture of the long-used "C89" standard which is the most familiar C dialect, corresponding closely to the classic Kernighan & Ritchie book we will be using, and a couple of features of C99 that were "back ported" from early C++, namely "// comments" and declaration at first use.

C++ was standardized in 1998 by the ANSI and ISO standards organizations (C++98), and again in 2011, 2014, 2017, and most recently in 2020 (C++11, C++14, C++17, C++20), where important additions were made to the language and the Standard Libraries. C++17 will be the Standard used in the gcc reference implementation for the course, which is essentially C++17's feature-complete. But mostly we will be using the features that have been available starting with C++11. The following rules apply: *This course will teach the specified Standard C and C++. Except when specifically stated, you must use only that specified Standard C and C++ in your projects. You may not use any 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 and currently supported by CAEN are gcc 9.1.0 C++17 under Linux, Apple's Xcode Integrated Development Environment (IDE) for Mac OS X, especially 10.x and later using the LLVM "Clang" compiler and a LLVM implementation of the Standard Library. For Windows, use the newest versions of MicroSoft Visual Studio C++, which has at least C++14 support. The project grading system runs under Linux, which is intended to be the same gcc 9.1.0 CAEN installation. You will have to use Unix/Linux to submit your projects; instructions will be provided, but you should have the basic Unix/Linux skills involved.

Debugger usage is required. In all of the programming environments, you should use a debugger right from the beginning. The value of this tool for serious programming cannot be overstated. If you haven't used a debugger, starting learning one and 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 — which generally have not improved since I (the Professor) was your age (really!).

Programming Projects

How to do well on the projects. There is a link 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 both important advice and further details on the course policies.

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 the course, you are expected to design and write high-quality code that makes full and appropriate use of the course material presented up to that point, even when the specifications do not specifically require it.

Project teams. Teams are *forbidden* on Projects 0-5 and are allowed only on the last project, Project 6, where forming a team is optional and formed by the students involved. Details will be provided in the Project 6 document. See also the discussion below on Academic Integrity.

Project grading. Projects will be graded in three ways: (1) 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 autograder credit, you must meet all three of these requirements.

(2) In *spot-check grading*, using both eyeballs and automated tools, I will check very briefly to see if required design tasks and components appear to be actually attempted, and whether serious and simple code quality problems are present. These issues are always clearly presented in the course material and the project specifications, and are easy to check. For example, it is specifically

explained that you should not use redundant code like `this->member_variable_name` to access member variables in a member function, and violations are easy to spot. There are vitally important rules for header files which can be checked quite easily. If the project requires that the copy-swap idiom or a Singleton pattern must be used, it is easy to check for these code "signatures". If problems appear in this spot-check, your autograder score will be reduced by a substantial amount; having the score reduced to zero is a possibility. These spot-checks are strong enough that sloppy programmers lose a lot of credit; all you have to do to avoid a disaster here is to pay attention to the course and project information and write the code properly — which gives you a chance to practice the concepts and techniques — the point of the course!

(3) Using *human grading* (my eyes and brain), I will evaluate your actual code for (a) *code quality*, which includes much more than just "style" (see the Coding Standards); (b) whether specified concepts and techniques were used; (c) whether course guidelines for quality design were followed; and (d) whether your design was well thought out and implemented. Because I physically mark up a hard copy of your code, the human grading is very laborious. But it is also the most useful feedback for becoming an advanced programmer. The human grading will be weighted much more heavily than the autograding scoring, and will only be done if the program is mostly working, as detailed below.

Seven projects are planned. The first six, Projects 0 - 5, will involve autograder grading. Project 6 will involve only human grading. Projects 1 and 3 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. This arrangement reflects the idea that a *minimum* requirement is that the program must run correctly and meet specifications on the component behaviors; if it meets this requirement, then the quality of the code is what is most important. There is no point in evaluating the quality of a program that does not mostly work, and a project that works but is badly written is not worth much.

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 (Projects 0, 2, 4, 5) will be subject to *spot-check* evaluation as described above.

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 and time — the project deadlines are "hard" — there will be no late submissions, no "late days," no "free late days," and no sloppiness about accepting a late project. *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. For projects 0-5, 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. I will announce a revised deadline if there are serious problems that are my fault or prevent me from making the autograder available on schedule or providing timely help. Do not mistake getting granting an extension for leniency in the deadlines or the grading. The rules are:

- Only the Professor can grant an extension, and it must be documented in email.
- Only documented medical problems or personal emergencies will be accepted 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.
- Granting an extension is giving you a new deadline for the project work. Once this is agreed and documented in email, then the new deadline will be strictly enforced just like the original deadline. See above about late project submissions.
- 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, spilled coffee into your computer, etc. You should have backed up your work! 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!). Especially notice that job interviews are not accepted as a reason for an extension or any other departure from the course rules. While doing well in the course might help you get a good job, getting a job is not part of this course's work! 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.

Extensions to team projects. The above extension policy applies to the individual team member seeking the extension, but if a team is involved (possible on Project 6 only) the situation is problematic — it is potentially unfair to other students that a whole team should get an extension because of one member's disability. To give the Professor a chance to resolve the situation in a way that is fair to everybody, the following rules apply:

- An extension given to one member of a team does ***not*** automatically apply to the whole team.
- Every member of a team must be fully informed of the situation. Therefore, when making a request for an extension, the team member with the problem must email the Professor *with a cc to all team members*; personal information can be sent just to the Professor.
- Once an extension has been requested by a team member, *all members of the team must immediately contact the Professor, with cc to other team members*, to verify that they have been notified. The Professor will then arrange a discussion with the whole team about how the situation will be handled.
- Unless the Professor has resolved the situation with the whole team, and documented it with email, no extension will be granted.
- If these rules have not been followed, any extension granted will be automatically (even retroactively) rescinded and canceled.

The situation will be handled in one of the following ways, in descending order of desirability:

- The team will redistribute the workload so that the normal project deadline can still be met. For example, if one person is unable to write or type, but is otherwise capable, that person can still contribute in a variety of ways, such as design discussions and critiques, being the watcher in pair programming, doing thorough quality-control checks of the code base against the Coding Standards, etc. These are not minor roles — doing this work well, can vastly improve the quality of the final project.
- The team will be disbanded, an individual extension will be granted for the team member with the problem, and the other team members will meet the normal deadline. Agreements will be made to share any work done before the problem arose.

Reading Assignments

Brief papers required. We are using books containing considerable practical and theoretical 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. This is essentially a "flipped classroom" except high-efficiency reading replaces watching a recorded lecture; your reading has a much higher information transfer rate for the basic content than listening to someone talk.

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*. You must be sure that your paper covers all assigned chapters, handouts, etc; a common error is to leave one of the items out, especially handouts. There are no other requirements for the content of the paper. The following are ways to meet this requirement that will also help you learn the material: You can list questions you have about the material; you can relate the material to your own interests or experiences; you can describe "what I didn't already know". It just has to be obvious that you read the entire assignment. Important: simply copy-pasting material from an electronic version of the book does not make it obvious that you read the material.

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. More pages are OK if it makes it easier to get this done. However, if you hand in more than one piece of paper, *you must **staple** them together*. Printed output on both sides of one sheet of paper, 10 or 12 pt Times, margins no more than 1 inch, 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. Electronic submission is normally not allowed because I work from a hard copy for speed; if electronic submission seems required for a special situation, you must get the Professor's 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 reading the material before class is important, the reading papers are part of 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 scored as -1 point. 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 as a significant part of the total course grade; thus the reward for keeping up on the readings is substantial. Skipping reading papers can be a real problem, 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 (with a "U"), 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 (even if you were not present), whereupon they will be scored as late if satisfactory.

If you return a paper as a resubmission, early submission, late submission, etc., please write at note at the top to let me know what is going on. This makes it easier for me to evaluate your paper properly and keep accurate records.

Grades and Grading

Getting the feedback. All graded materials will be returned during a subsequent class period, or during subsequent office hours. If you were absent when a reading paper, project, or exam was returned in class, I expect you to ask for it — I take a dim view of students who do not bother to collect the available feedback while the course is in progress.

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 when I fail to follow my own grading policy or specifications correctly. If I mis-grade 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, for example, within a week of when exams were handed back in class to the students who were there.

Course grade determination. The course programming projects make up 58% of your course grade; seven projects are planned with the first six contributing 8% each and the last counting for 10%. Any modifications to the number and weight of projects will be announced. The course exams constitute 36% of your final course grade, 16% for the midterm exam and 20% for the final exam. The reading assignment papers constitute the remaining 6% 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 (A's or B'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 item score, such as the 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 means that at least some A's will be awarded, but does not require any grades below A, nor does it require any failing grades! However, you still have to work hard to get a passing grade, 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 at any time in the course.

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 — the Professor's decisions on such borderline cases are final and not a subject of negotiation. In no case will you be awarded a lower grade than your scores indicate.

Consistent with University grading policy, Incomplete grades will be given only rarely, 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. There is no provision for students to do any kind of additional or "extra credit" work to avoid a bad grade, so you need to focus on the assigned work during the course to earn your grade.

If you are dissatisfied with your grade, be informed that: *Your grade is not a subject of negotiation*. My 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 excellent 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*. *Only Project 6 allows teams*. 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 Professor or GSI 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 unsure 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, the Professor 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 GSI is not allowed. Not only will you benefit more by thinking through the issues yourself, but even in cases where students have exchanged just "ideas" about design, a common result is that the "ideas" are pretty bad — your fellow students in this course are generally as ignorant as you are! This prohibition also applies to tutors, regardless of their source or status. The only allowed sources for specific design discussions 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, either when working alone or when collaboration is permitted, your code must not be publicly accessible. Notice that github provides private acounts for students.

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. In other words, sharing "ideas" is OK, but the specifics of the design and code must be your own. (But beware: somebody else's "idea" can be lousy; think it through yourself!)
- 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 (if it disagrees with my materials or advice, please contact me before using it). 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 directly residing on the course web site. ***Note***: You are still responsible for the correctness of your program; an 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, code not directly residing on the course website (even if it is linked to from a course web page), web sites about programming, 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 schools).

- In case of doubt, *you are required to ask the Professor* 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*.