Com S 327
Advanced Programming Techniques
Spring 2024

Updated: 13:46 Friday 19th January, 2024

Instructor

Instructor	Jeremy Sheaffer
Office	1200E Communications
Email	sheaffer@iastate.edu (please preface all email subjects with "CS 327: ")
Lecture	TR 9:30–10:45 Lagomarcino 1155 (Section 1)
	TR 12:40–1:55 Lagomarcino 1155 (Section 2)
Office Hours	MT 10:55-12:25

Note: Email is *not* a preferred method of communication. Instead please use Piazza (you can send private messages to me there) or talk to me after lecture or in office hours.

Teaching Assistants

Name	Email	Office Hours				Location
Farjana Sultana Samia	fssamia@iastate.edu		T	R	14:00–16:00	Pearson 0112
Asif Siddique	asif64@iastate.edu	M		F	14:00–16:00	Pearson 0112
Yongyun Song	yongyun@iastate.edu	M	T	F	16:00–18:00	Pearson 0112

Webex/Zoom meetings are available by arrangement during scheduled office hours. Reach out to the appropriate instructor. This is intended for sick or immunocompromised students; please do not abuse it.

Course Summary (from the catalog)

Object-oriented programming experience using a language suitable for exploring advanced topics in programming. Topics include memory management, parameter passing, inheritance, compiling, debugging, and maintaining programs. Significant programming projects.

Course Summary (from the instructor)

We will cover the topics described above with two languages, C and C++, used as drivers. We will spend roughly half of the course on C and half on C++. We will also devote some time to programming environment, build tools, version control, and debugging; these are all concepts that you will need to understand and employ to be successful in this course and beyond.

Course Objectives

After successfully completing this course, students will:

- understand differences between managed languages (e.g., Java) and unmanaged languages (e.g., C and C++);
- understand and be able to use simple build systems (e.g., make);
- be able to design and build large programs from specification;
- be able to use third-party libraries in programs;
- be able to read, write, and modify C programs;
- understand memory management techniques for C;
- be able to read, write, and modify C++ programs;
- understand memory management techniques for C++;
- understand C++ templates and the standard template library.

Course Outcomes

This course has three major ABET outcomes:

1. By the end of this course, students will be able to produce efficient and correct C and C++ programs of significant lengths from specifications.
2. By the end of this course, students will be able to understand and use advanced C and C++ features in software development and maintenance.
3. By the end of this course, students will be able to write and debug large C and C++ programs based on English descriptions or pseudocode.

Prerequisites

Course prerequisites are Com S 228 and Math 165, or instructor permission. No exceptions. If you do not meet these prerequisites, it is the policy of the Computer Science department that your materials will not be graded.

Textbooks

There are no required texts for this course. All required material will be presented in lecture. Supplemental material may be found in the following excellent texts:

- Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language*.
- Bjarne Stroustrup, *The C++ Programming Language*.
- Erich Gamma, John Vlissides, Ralph Johnson, and Richard Helm, *Design Patterns: Elements of Reusable Object-Oriented Software*.

Recommended Software

In the past, I have recommended that students work directly on the Computer Science department's student server, `pyrite.cs.iastate.edu`. While this is still a good solution, there are some drawbacks, namely that it requires a constant network connection, which may get laggy, and that the server sometimes experiences heavy loads. Since the Spring 2016 semester we have been using virtual machines with great success. Using a Linux VM (or a native Linux computer) has been the recommended working solution ever since.

Linux VMs

I have created an Ubuntu Linux 20.10 image for the use of students in this class. I have installed the drivers that allow Linux to interface with the host operating system through the VM, so you should be able to configure shared directories, cut-and-paste between host and client, and other, similar convenient things. I don't actually know how to do these things, so you'll need to read the documentation or google to get that functionality configured (if you want it).

I have done a very small amount of configuration on these installations, including installing Emacs (the One True Editor), Valgrind (a memory debugging tool that you will want), Google Chrome, and compiler updates to the current version of GCC, GCC 10.2.0.

I have created a user account, `student` with password `student` and did a small amount of configuration of that account.

The image is available on Cybox at the following link:

https://iastate.box.com/s/2ol9mgpmduvhfq07zc2tfybo56awtelj

This file is moderately large, slightly over 7 GB. It is a *VirtualBox* file. You will need to install VirtualBox (https://www.virtualbox.org/wiki/Downloads) and import the images (File >> Import Appliance). Once you have successfully imported, you can delete the OVA file. The system is configured to use 1GB of memory; in retrospect, this isn't much and should probably be increased. The virtual hard drive will grow as needed up to a maximum of 32 GB. Much of the configuration can be changed, but only if the machine is not actually running.

Remote Server

It is always possible to complete the programming assignments by working directly on `pyrite.cs.iastate.edu` or on any of the Linux machines in the labs. Connect to `pyrite` using ssh. A nice, free Windows ssh client is PuTTY (http://www.putty.org/). To resolve `pyrite` from off campus, you will need to first connect to the Iowa State VPN (https://vpn.iastate.edu/).

Other Necessities

In a Linux environment, you can edit with Emacs, vi, or Pico (the first two are powerful and used by professional programmers all over the world, while the latter is simple and used by undergraduate computer science students with tunnel vision all over the world).

Regardless of your working environment, you will need a C and a C++ compiler, an editor, and a debugger. You may use whatever specific tools you like, so long as your submitted code compiles and runs on the VM. In lecture, I will use GCC and G++ for compiling, Emacs for editing, and GDB for debugging, because these are what I use for real programming.

Be careful if you use Visual Studio! Visual Studio will often compile code that does not compile under GCC. If you are not developing on `pyrite` or one of the VMs, be sure to give yourself enough time for testing before you must submit.

The main project will use the ncurses library, which may be obtained in many different ways. Official site: http://www.gnu.org/software/ncurses/ncurses.html.

All submissions must compile with GNU Make for credit. Official site: http://www.gnu.org/software/make/

Helpful Extras

Version control software, such as `git` or `subversion`. Among other things, this allows you to keep your source code synchronized between your personal machine and the testing machine.

- Subversion book: http://svnbook.red-bean.com/
- Git book: http://git-scm.com/book

`Valgrind` (http://valgrind.org/) or some other memory profiling tool. This can save *hours* of work tracking down segmentation faults.

If you are running Linux, you probably have all of this already. Windows users can get everything by installing Cygwin (https://www.cygwin.com/), but it is easy to botch the installation; it is probably easier to install VirtualBox (https://www.virtualbox.org/wiki/Downloads) and then install a full Linux distribution on top of it. Mac OSX used to ship with GCC. My understanding is that it no longer does. Mac users: Google is your friend.

I cannot provide you with technical support on any platform save Linux. I have almost no experience with systems that are not either UNIX-based or embedded. My experience teaching this class in the past suggests that students who insist on working in Windows because they are more comfortable there struggle the entire semester, while those who commit to working in Linux struggle only at the beginning.

Lecture Attendance

Attendance is mandatory. You are responsible for all material presented in lecture.

Canvas and Piazza

We will be using Canvas only for the gradebook and homework submission. All other course materials, including announcements and discussion forums will be managed on Piazza.

Course Work

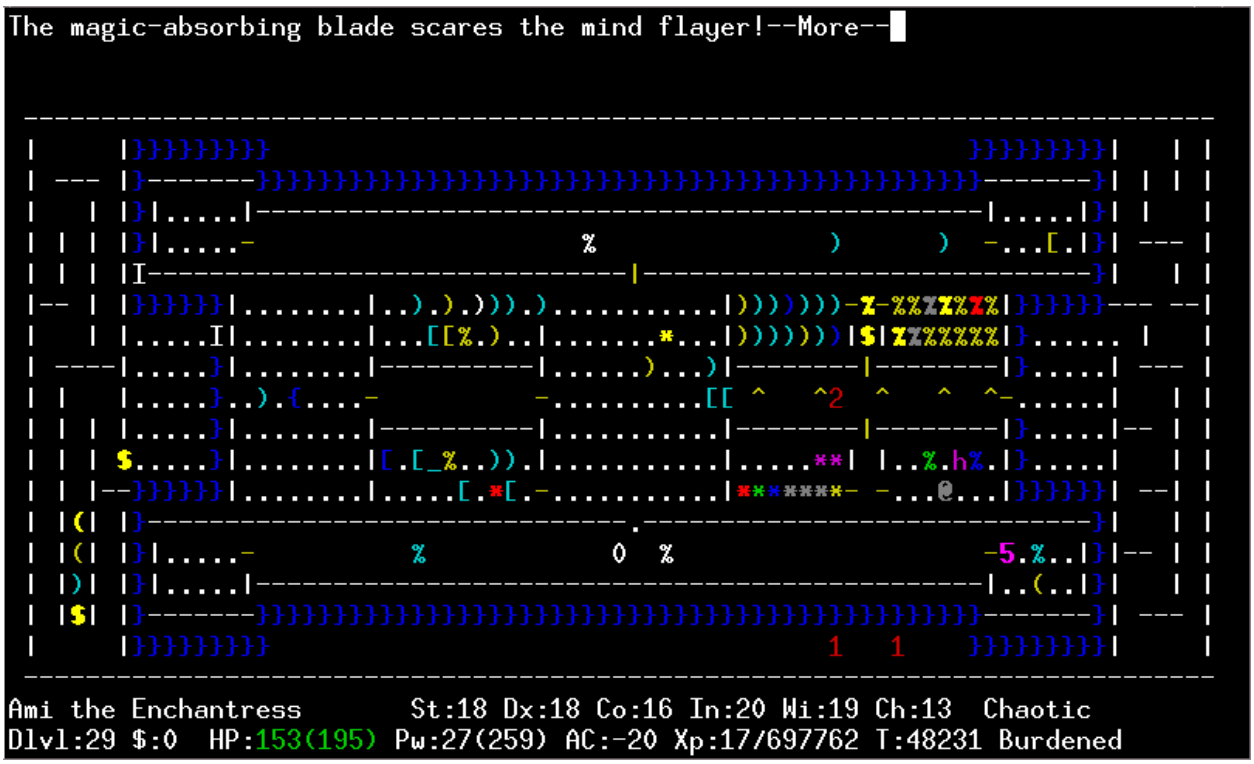
Projects (60%)

Over the course of the semester, we will be building a Pokémon-inspired Roguelike game using ASCII graphics. Two iconic modern examples of roguelike games are Angband and

NetHack, and (most of) you are of an age that I won't have to tell you about Pokémon. The figure shows a screen shot of a typical NetHack game. Your games will probably not be nearly as extensive as these two examples—unless you want them to be!—but by the end of the semester, you will have developed a playable game with, at minimum, nice-looking, randomly-generated dungeon maps, items and monsters that are specified in files that your classmates' games will be able to read and use, turn-based play with keyboard input and terminal output, and the ability to save and restore. Many people write such games in a week as a part of a contest, for fun; some of them are even pretty good.

Certain functionality will be due, beginning the second week of classes, every week except for Fall Break and Dead Week at 10:00pm on Wednesdays. Project requirements will be specified in advance, so that you may work ahead, but at any given time, you will only be graded on the functionality required by that date.

All assignments must be turned in with an ASCII readme file, named `README` and containing no more than 150 words, which describes your project's compliance with the specification to date including relevant function, data structures, and file names. Assignments must also contain an ASCII change log file named `CHANGELOG` that contains a cumulative list of dated updates from day 1 (e.g., "started project"). The change log will list every significant change you make to your project, including bug discoveries, bug fixes, functionality additions and changes, etc.



Screenshot from a game of NetHack, showing a message on top (the player has attacked a mind flayer, a very dangerous monster that will suck out your brains), dungeon view in the middle, including the character (the '`@`'), the mind flayer (the '`h`'), gold ('`$`'), various monsters that the character is aware of magically (the numbers, which denote their approximate strengths), a moat (the many '}'), many weapons, pieces of armor, gems, traps, doors, corridors, and items of food, and, on the bottom, the character's status.

All projects must include a working makefile, named `Makefile`, which correctly builds a binary with debugging symbols on `pyrite`. We will discuss these things in the first week of class.

All projects must be submitted via the appropriate assignment page on Canvas. Projects should be turned in as a *tarball* (a gzipped tar archive). Basic instructions, with an example, on how to create and extract acceptable tarballs follow. Tarballs are to be named

`<surname>_<forename>.assignment-<assignment number>.tar.gz`

and should extract to a directory named

`<surname>_<forename>.assignment-<assignment number>`

where `<surname>` is your family name, `<forename>` is your given name, `<assignment number>` is the submitted assignment number, and all other characters are literals. Thus, if *I* am submitting assignment 1.05, my tarball will be named

`sheaffer_jeremy.assignment-1.05.tar.gz`

and it will expand to the directory

`sheaffer_jeremy.assignment-1.05`

which will contain all of my source files, and `Makefile`, `README`, and `CHANGELOG` as defined above, but no generated files (run `make clean` first!).

Still using assignment 1.05 as the driving example, the simplest way to create a tarball follows. This is an actual session from my shell, saved with a program named `script`. I have added comments inline, preceding them with '`%%` ', which are not part of the actually shell I/O. Lines beginning with `sheaffer@sheaffer...` are input. Lines ending with a backslash ('\') are continued on the following line. I've indented continued lines by two spaces to make them stand out. All other lines are output.

```
Script started on Thu 08 Jan 2015 08:20:53 AM CST
%% Run 'make clean' to get rid of the binaries
sheaffer@sheaffer-e7440:~/classes/semester/s2015/229/assignments/assignment1.05$ make clean
rm -f dungeon.o cds/heap.o rlg229
%% Move into the parent directory so that we can make a tarball of the
%% current directory
sheaffer@sheaffer-e7440:~/classes/semester/s2015/229/assignments/assignment1.05$ cd ..
%% My directory isn't named as per the specification, so create a copy of it.
%% '-R' makes it copy recursively (a deep copy) so the 'cds' subdirectory is
%% copied, too, or you could rename it with 'mv'.
sheaffer@sheaffer-e7440:~/classes/semester/s2015/229/assignments$ cp -R assignment1.05/ \
  sheaffer_jeremy.assignment-1.05
%% Create the tarball.  Four switches to tar.  'c' means create, 'v' means
%% verbose, 'f' means force, and 'z' means compress with gzip.  'v' and 'f'
%% aren't strictly needed, but 'v' does cause it to give the file list that
%% follows.
%% Note that the backslash ending the next line is to escape the newline in
%% a multi-line command.  It is not part of the command!
sheaffer@sheaffer-e7440:~/classes/semester/s2015/229/assignments$ tar cvfz \
  sheaffer_jeremy.assignment-1.05.tar.gz sheaffer_jeremy.assignment-1.05/
sheaffer_jeremy.assignment-1.05/
sheaffer_jeremy.assignment-1.05/cds/
sheaffer_jeremy.assignment-1.05/cds/hash.h
sheaffer_jeremy.assignment-1.05/cds/tree.c
sheaffer_jeremy.assignment-1.05/cds/macros.h
sheaffer_jeremy.assignment-1.05/cds/neural_net.h
sheaffer_jeremy.assignment-1.05/cds/neural_net.c
sheaffer_jeremy.assignment-1.05/cds/list.c
sheaffer_jeremy.assignment-1.05/cds/heap.c
sheaffer_jeremy.assignment-1.05/cds/list.h
sheaffer_jeremy.assignment-1.05/cds/spinlock.h
sheaffer_jeremy.assignment-1.05/cds/heap.h
sheaffer_jeremy.assignment-1.05/cds/hash.c
sheaffer_jeremy.assignment-1.05/cds/heap.d
sheaffer_jeremy.assignment-1.05/cds/circular_queue.h
sheaffer_jeremy.assignment-1.05/cds/circular_queue.c
sheaffer_jeremy.assignment-1.05/cds/tree.h
sheaffer_jeremy.assignment-1.05/Makefile
sheaffer_jeremy.assignment-1.05/dungeon.h
sheaffer_jeremy.assignment-1.05/dungeon.c
sheaffer_jeremy.assignment-1.05/CHANGELOG
sheaffer_jeremy.assignment-1.05/README
%% At this point, I am done.  I could submit, however, I don't want to risk a
%% penalty to my grade, so I am going to test it out.  I can't really expand
%% the archive here, because the target directory exists.  Since that
%% directory is a copy, I could just delete it now, but if I made a mistake
%% and need to try again, I would rather not have to retype all that stuff, so
%% let's make a temporary directory to expand in, call it 'test'.
```

```
sheaffer@sheaffer-e7440:~/classes/semester/s2015/229/assignments$ mkdir test
%% And move into it...
sheaffer@sheaffer-e7440:~/classes/semester/s2015/229/assignments$ cd test
%% Now to open up the archive.  gzip compresses and decompresses.  The 'd'
%% switch tells it to decompress.  'c' tells it to concatenate the output to
%% the standard output, then we pipe ('|') that to tar, where 'x' means to
%% expand ('v' and 'f' are as above) and a file name of '-' means to read the
%% file from the standard input.  I can see in the output that follows that
%% the files are all there.  I can move in, do a build, some testing, etc.,
%% as I please.
sheaffer@sheaffer-e7440:~/classes/semester/s2015/229/assignments/test$ gzip -dc \
  ../sheaffer_jeremy.assignment-1.05.tar.gz |tar xvf -
sheaffer_jeremy.assignment-1.05/
sheaffer_jeremy.assignment-1.05/cds/
sheaffer_jeremy.assignment-1.05/cds/hash.h
sheaffer_jeremy.assignment-1.05/cds/tree.c
sheaffer_jeremy.assignment-1.05/cds/macros.h
sheaffer_jeremy.assignment-1.05/cds/neural_net.h
sheaffer_jeremy.assignment-1.05/cds/neural_net.c
 sheaffer_jeremy.assignment-1.05/cds/list.c
sheaffer_jeremy.assignment-1.05/cds/heap.c
sheaffer_jeremy.assignment-1.05/cds/list.h
sheaffer_jeremy.assignment-1.05/cds/spinlock.h
sheaffer_jeremy.assignment-1.05/cds/heap.h
sheaffer_jeremy.assignment-1.05/cds/hash.c
sheaffer_jeremy.assignment-1.05/cds/heap.d
sheaffer_jeremy.assignment-1.05/cds/circular_queue.h
sheaffer_jeremy.assignment-1.05/cds/circular_queue.c
sheaffer_jeremy.assignment-1.05/cds/tree.h
sheaffer_jeremy.assignment-1.05/Makefile
sheaffer_jeremy.assignment-1.05/dungeon.h
sheaffer_jeremy.assignment-1.05/dungeon.c
sheaffer_jeremy.assignment-1.05/CHANGELOG
sheaffer_jeremy.assignment-1.05/README
sheaffer@sheaffer-e7440:~/classes/semester/s2015/229/assignments/test$ cd ..
%% Looks good!  I am ready to submit.  I don't need the temporary directory,
%% nor the renamed directory, so I am cleaning them up.
sheaffer@sheaffer-e7440:~/classes/semester/s2015/229/assignments$ rm -rf test
sheaffer_jeremy.assignment-1.05
%% And exiting the 'script' program.
sheaffer@sheaffer-e7440:~/classes/semester/s2015/229/assignments$ exit

Script done on Thu 08 Jan 2015 08:22:54 AM CST
```

Given the cumulative nature of the assignments, we will attempt to return all assignments, except assignment 12, within 48 hours. **Assignments submitted with formats that do not match the above specification will receive a penalty of at least 20%. A assignment without an up-to-date readme, an up-to-date changelog, and a working Makefile will not be graded and will receive a grade of 0. A required feature which is not discussed in the readme will not be evaluated, and thus will not receive credit.** Since the assignment is cumulative, it will grow to such a size that searching your code for required features will become intractable to the graders. These rules are necessary to make grading possible.

Because the assignments are cumulative and we do not want students to fall behind with no reasonable chance of catching up, the instructor will periodically release code checkpoints to the class which may be used in part or in whole by any student in the class in subsequent submissions.

The first assignment is special, intended to get you acquainted with the development environment and to provide a

gentle introduction to C, and it is not part of your game. For this reason, we will start numbering with 0.

Assignment submission schedule

Assignment	Due Date
Assignment 0	`Wed Jan 24 22:00:00 CST 2023`
Assignment 1.01	`Wed Jan 31 22:00:00 CST 2023`
Assignment 1.02	`Wed Feb  7 22:00:00 CST 2023`
Assignment 1.03	`Wed Feb 14 22:00:00 CST 2023`
Assignment 1.04	`Wed Feb 21 22:00:00 CST 2023`
Assignment 1.05	`Wed Feb 28 22:00:00 CST 2023`
Assignment 1.06	`Wed Mar  6 22:00:00 CST 2023`
Assignment 1.07	`Wed Mar 20 22:00:00 CDT 2023`
Assignment 1.08	`Wed Mar 27 22:00:00 CDT 2023`
Assignment 1.09	`Wed Apr  3 22:00:00 CDT 2023`
Assignment 1.10	`Wed Apr 10 22:00:00 CDT 2023`
Assignment 1.11	`Wed Apr 17 22:00:00 CDT 2023`
Assignment 1.12/2	`Wed Apr 24 22:00:00 CDT 2023`

All assignments will be graded out of 10 points. Some assignments may be extended, reducing the total number of assignments; thus, the final weight of each assignment will be 55% divided by the final number of assignments.

Exams (40%)

There will be two exams, a midterm and a final.

Exam schedule

Exam	Time	Location	Weight
Midterm	Thu Mar 8 lecture meeting	Regular lecture location	20%
Final (Section 1)	Tue May 7 9:45-11:45 am	Regular lecture location	20%
Final (Section 2)	Thu May 9 9:45-11:45 am	Regular lecture location	20%

You are responsible for providing at least 1 week advanced notice if you have an exam conflict. Students who give sufficient notice will be permitted to schedule an alternate exam time.

The class meeting before each exam will include review of a previous exam. The class meeting after each exam will include review of your exam.

Lateness Policy

Late work will be accepted with no penalty until such time as the TAs begin grading, at which time submission will be closed and no further submissions will be accepted for the assignment. We cannot say when TAs will begin grading, but given that we intend to return work within 48 hours, it could be very soon after the deadline. Thus the optimal submission strategy to maximize your grade is to complete and submit the required functionality by the deadline. Except for projects 0 and 1.12, because it is cumulative, all work benefits the project as a whole, even if it occurs after its respective deadline, so work is never wasted. If you fail to have a complete, correct submission by the deadline, the optimal remaining strategy is to continue to develop, and submit regularly, until such time as you complete the assignment or you find submission closed. The TAs will always take the last submission that was available at the time that they download the projects.

Grading

Midterm and final grades will be curved. The curves will not be determined until grades are calculated, and will be designed to fairly rank your performance against that of your peers and reward you appropriately for your work. A traditional 10-point scale establishes a lower bound on your grade now. Any curve that is applied at the midterm will establish a new lower bound for the final grade, which may in turn be curved even more in your favor. Either or both of the midterm and final curves may be null.

Bug Bounties

All solution code that is released by the instructor to Canvas is subject to a bug bounty. The bug bounty works as follows:

Solutions may contain a `BUGS` file listing known bugs. These bugs are not eligible for bug bounties.

When you discover what you believe to be a bug in solution code, you must:

- find an input that reproduces that bug;
- write a fix for the bug which is itself (apparently) bug free;
- use `diff` to produce a *patch* that will patch the solution to your bug-fixed version[1]; and
- create a Piazza post to "Entire Class" (a public post) in the "bug bounties" folder with a subject that indicates the assignment number containing a description of the bug, how to reproduce, and your patch file. *Do not* submit your solution sources, only the patch file.

If you are the first to post a correct bug fix, you will be rewarded with an increase of one grade level (that is, A- becomes A, C+ becomes B-, etc., but only kudos if you already have an A) to your final course grade.

Just because something doesn't work the way you think it should doesn't make it a bug. I reserve the right to reject your submission if it "repairs" something that I consider not a bug. You may ask me before you work if I would consider X to be a bug; I will give you the best answer I can given the accuracy and detail of your question.

Be advised that if you post an incorrect bug report, input, or fix, the information immediately becomes available to your classmates, who may then beat you with the correction, so try to get it right the first time.

No bug bounties will be accepted after 22:00 the Friday before Dead Week. No bug bounties will be evaluated before dead week (i.e., you will receive no feedback from me regarding your bug report until—at earliest—Monday of Dead Week).

Academic Integrity

Exams and any and all required software functionality are to be your own work. We will take academic dishonesty seriously. We will report it to the Dean of Students. If you are found responsible for academic dishonesty in this class, you will receive a failing mark for the course. We believe that you are mature enough to understand "integrity" without a great deal of detail in the syllabus. Here we list some examples of things which you may and may not do in this class. If you are ever unsure, discuss it with the instructor.

The following examples should be taken in spirit, not as a complete list. For example, you may not edit another student's code, so it should be clear that you also may not write code on paper for another student to use. With respect to programming assignments, you may not:

- Share required elements of your project source code with anyone before the instructor clears those elements for sharing.
- Study required elements of another students source code.

[1] You may not know the keywords to google this, so here's a link that explains it: https://stackoverflow.com/questions/9980186/how-to-create-a-patch-for-a-whole-directory-to-update-it

- Submit another student's work as your own.
- Debug another student's source code.
- Edit another student's code.
- Use code from the Web or elsewhere to satisfy required functionality in your project before submission for that functionality is closed.
- Use third party code in your project without attribution.

You may:

- Discuss algorithms, data structures, and design with other students and on Canvas
- Directly use third-party code for non-required functionality with complete and correct attribution
- Directly use third-party code for required functionality, provided submission for that functionality has already been closed and you completely and correctly attribute the author.
- Implement algorithms learned from third-party sources in your code
- Share code that implements non-required functionality on Canvas.
- Share code that implements required functionality on Canvas after the instructor gives clearance to do so.
- Help another student debug, so long as that student "drives."

The University's policies on academic dishonesty are located online at:

https://www.studentconduct.dso.iastate.edu/know-the-code-resources/resources-for-students/academic-misconduct.

Please speak with the instructor directly if you have any questions or doubts.

Disabilities

Iowa State University complies with the Americans with Disabilities Act and Section 504 of the Rehabilitation Act. Any student who may require an accommodation under such provisions should contact the instructor as soon as possible and no later than the end of the first week of class or as soon as you become aware. Please obtain a SAAR form verifying your disability and specifying the accommodation you will need. No retrospective accommodations will be required in this class.

Harassment and Discrimination

Iowa State University strives to maintain our campus as a place of work and study for faculty, staff, and students that is free of all forms of prohibited discrimination and harassment based upon race, ethnicity, sex (including sexual assault), pregnancy, color, religion, national origin, physical or mental disability, age, marital status, sexual orientation, gender identity, genetic information, or status as a U.S. veteran. Any student who has concerns about such behavior should contact his or her instructor, Student Assistance at 515-294-1020 or email dso-sas@iastate.edu, or the Office of Equal Opportunity and Compliance at 515-294-7612.

Religious Accommodation

If an academic or work requirement conflicts with your religious practices or observances, you may request reasonable accommodations. Your request must be in writing, and your instructor or supervisor will review the request. You or your instructor may also seek assistance from the Dean of Students Office or the Office of Equal Opportunity and Compliance.