# Preparing and Submitting Assignments

Steven J. Zeil

August 20, 2013

## Contents

This document explains the procedures for turning in assignments in this course.

# 1 Programming Assignments

**Why are the Programs used in Assignments so Big?** In one sense, Object-Oriented programming is all about organizing the abstract data types that make up a program. A program that's small enough to need only a few abstract data types doesn't need OO, won't benefit from it, and can't be used to practice OO. That creates a bit of a dilemma for an instructor when it comes to composing assignments. If you're really supposed to get a feel for how OO works, it can't be on programs that are small enough for you to write in a week or so.

Most of the assignments in this course will therefore involve making small changes to a set of code for a nearly complete program, or altering a completely working program to take advantage of some new data structure or algorithm we have studied. As it happens, that's probably a more realistic approach. Most real-world programming projects will not involve creating an entire program from scratch. Far more often, you will be asked to modify, fix, or enhance existing code.

You yourself will probably not be writing that much code in any given assignment, But part of the challenge in working with programs at this scale is that you have to develop a knack for reading code and seeing where you need to make changes and what you need to leave alone.

Grading is performed on the CS Dept's Linux servers. You may do your development work on the CS Dept's PCs, or on other machines more convenient to you. It's a good idea, though, before you submit your work, to upload your code to the CS Unix network and check to be sure that it still works after doing so. This is not something you should take for granted!

**The Grading Criteria for Programming Assignments**

1. Satisfy the stipulations and limitations of the assignment.

   Each assignment will have certain stipulations. You may be limited in what files you can change, what data structures you can use, etc. The point of any assignment is to practice and demonstrate your ability with certain skills and techniques that we have covered in class.

   You might be able to get that program running in some completely different manner, maybe rewriting the whole thing from scratch. It might even be easier for you to do it that way. That doesn't matter. If you have not employed the stipulated techniques, your submission will be disqualified.

2. Produce the correct output.

   The primary scoring criteria for programming projects is the percentage of the test cases on which your code produces correct output.

   While it's common in beginning programming classes to write interactive code that prompts for inputs and accepts them from the keyboard, few programs these days work that way. Most truly interactive programs will work through a graphic user interface. And we will do those late in the semester. But there *are* still many programs that work through simple text input/output interfaces.

These programs typically read and write to files or possibly directly to other programs. Consequently, *the I/O formats of these programs are crucial and must be followed very precisely*.

**The Anti-Grading Criteria for Programming Assignments** These are things that you will *not* get points for in the grading of programming assignments:

1. The code is "almost correct" or "looks similar to" a correct solution.

   Sorry, but almost everyone who submits buggy code thinks that their code is "almost correct", that it "looks like" code that "should work". It's a well-known truism in our field that most software projects are 80% complete for 90% of their development time. It's very easy to get code to a point where it looks like something that should be correct and yet fails every test case submitted to it. The real effort in programming, and what you are rewarded for in the grading, is for getting past that point to something that actually passes one or more tests.

   This does not mean, as some folks complain, that I don't give partial credit on programming assignments. Some credit is awarded for code that compiles. More importantly, I generally have multiple test cases, and you will get credit for those that you pass, even if you fail some others.

2. Time spent on the assignment

   "I spent so many hours on this assignment." That's gratifying, but do you think that your classmates did not? Or, if someone managed to produce correct output while spending only half the time that you did, do you think that I should take points away from them for having done it more efficiently than you?

   Telling me how much time you spent on an assignment may earn you sympathy. And in some cases I may want to talk with you about whether you are using your time in a smart way. I may be able to suggest more efficient ways for you to approach your coding, testing, and debugging. But it does not demonstrate your mastery of whatever techniques the assignment is trying to showcase.

**The Auto-Grader** Most programs in this course will be graded automatically. You will submit your assignments electronically. Within a limited amount of time (typically 15-30 minutes, but this may vary depending upon how many other people are submitting) your code will be compiled. If it compiles successfully, it will be run on the instructor's test data and scored on how well it performs. The results of the compilation and testing will be emailed to you (at your CS email account) and will be available on the website.

The point of the auto-grader is to provide you quick feedback. In many cases, you may be able to use that feedback to fix things before the final due date of the assignment. If I graded in the "traditional" manner (you turn things in and I grade them after the due date), you would not get that opportunity. That can be very frustrating (to me as well as to the students) when students make "silly" mistakes such as submitting the wrong file, writing code that takes its input from the wrong source, writing output in an incorrect format, etc.

Some people complain that the auto-grader is too picky or too strict. That's a bit misguided. The auto-grader only runs the tests that I would have run if I were executing your program "manually". (Now, if you want to complain that *I* am too picky, that's another matter entirely.)

**Multiple Submissions** *If there is time before the due date, programs may be resubmitted without penalty at most twice (after the first submission, for a maximum of three total submissions).* A few programming assignments may not permit re-submissions, in which case that will be stated explicitly as part of the assignment description. Non-programming assignments, in general, cannot be resubmitted.

The reason for allowing resubmissions is to give you a chance to recover from "silly mistakes" as described earlier or from fundamental misunderstandings about what the program is supposed to be doing. It is not intended to allow you to keep chasing points by passing one more test each time. You are expected to test your own code before submitting, and, if you have done that, you should not actually be surprised by the percentage of my tests that you pass or fail. Now, if you did not make a silly mistake and have time left before the due date and can incrementally improve your percentage of tests passed, I'm not going to complain if you use the allowed resubmissions for that purpose. But I'm also not going to be sympathetic to requests for "just one more submission" in order for you to increase your percentage of tests passed.

**Appeals** Instructors are fallible too, and sometimes we will have a bad test. If you discover a bad test when you review the assignment solution, you can ask to have it re-scored.

If you have done multiple submissions and actually made your code score lower on your final submission than it had on an earlier one, you may ask that your score be rolled back to the higher score. You have to ask, though. I generally don't notice when this has happened unless you tell me.

On the other hand, i

## 1.1 Checking Your Code

### 1.1.1 Compiling

The official C++ compiler for this course is the Free Software Foundation's GNU g++ compiler, version 4.5. This compiler is available on the CS Dept's Linux workstations and Vista PCs (as the Code::Blocks package). You can also download this compiler for free and install it on your own PC.

Do *not* use Dev-C++, even if you used it in a prior class. Its version of the GNU g++ compiler is outdated.

Do *not* use the Microsoft Visual compiler. It has numerous bugs that make it incompatible with much of the code that will be used in this course.

It's important that you not only use the same compiler that we will use to grade your code, but that you compile your code the same way we will be compiling it - using the same commands and options. This will be easy for you to check. Each assignment will be supplied with a `makefile`. You can compile your code on a Linux system via the command

```
make
```

.

### 1.1.2 Executing

In most cases, the assignment description will provide details on how the program should be executed (i.e., what command line parameters to supply, where to find inputs and outputs). You may also find details on this provided in comments in the .cpp file containing the `main` function.

Most assignments will describe a sample input and its expected output. These may be provided as sample files in the assignment directory.

### 1.1.3 Testing

*You are responsible for testing your code before submitting it.* If you do a poor job of this, you should not be surprised if my tests find bugs that you were unaware of.

- With that said, my tests are seldom particularly subtle or difficult. Most of them follow the basic forms of black-box testing with which you should already be familiar. In some cases, you may need to write your own test drivers or other scaffolding.
- How can you tell if your output is correct? In most assignments, I will supply you with a compiled version of my own solution. You can run your program and mine on the same input and compare the outputs in head-to-head testing. Consequently, there should be very little question as to what output is expected for any given test input.
  - You will find these compiled versions of my solution in the bin directory of the assignment. Within that directory, you will find directories `Linux` and `Windows`, each containing an executable program.
    - Be sure that you choose the version appropriate to the machine you are running it on. You cannot execute a program compiled for Windows on a Linux machine or vice-versa!
    - The Linux versions are compiled for our CS Linux (Ubuntu) servers, available at `linux.cs.odu.edu`. They cannot be run under most other Unix variants. *Stay away from the Solaris ("fast") machines.* They'll just cause you problems in this course.
  - Note also that you do not need to copy the Linux program to your own directory in order to run it. (Doing so may actually cause confusion by overwriting the compiled executable from your own code.) Just run it via its full path:

    ```
    <b>~zeil/cs361/Assignments/assignment/bin/Linux/executableName any-arguments-required</b>
    ```

    And remember, when typing long path names in Linux shells, the Tab key can be used to auto-complete a file or directory name after you give the first few letters.
- Finally, do not trust that because your code works on a PC that it will work on Linux as well. Among other potential problems, my own experience is that Windows is more likely to provide newly allocated memory filled with zeros. This tends to hide errors involving uninitialized variables, missing return statements, and some pointer manipulation errors.

### 1.1.4 Submitting

Each assignment page will have a button or a link that you can use to submit your solution. You can upload files either from file system the machine at which you are browsing or directly from your CS Unix account.

The submissions page will also allow you to check to see if a prior submission has been graded and to view the most recent grade report. In most cases, you should have received this report by e-mail, but this provides you a backup in case the email message disappears into the Internet ether.

### 1.1.5 Afterwards

When you have completed an assignment, you can return to the submissions page and ask to see the solution. You will see not only my solution to the assignment but also the tests that I used for the evaluation. These tests are sometimes simple input files. Other times they are shell scripts that can be executed to run the test.

Even if (maybe especially if) you do poorly on an assignment, you can still learn from it by reviewing the solution. I assume that everyone does so. I'm happy to answer questions about assignment solutions, but if you do not ask, I will assume that you read the solution and understood it.

## 1.2 Files Provided in Assignment Directories

Each programming assignment is found in a separate directory, specified on the assignment web page. You will need to copy the files from that directory into your own working directory.

Make sure your files are protected! It is your responsibility to protect your assignments and other course work from access by your fellow students. Of course, they shouldn't be looking there, but the ODU honor code recognizes that making your work freely available for copying is also unethical behavior.

*Leaving your assignment solution files unprotected for your classmates to read is considered a violation of the ODU honor code and may result in sanctions up to and including expulsion from the University.*

Typically, each assignment directory will include a number of .cpp and .h files. Some of these are for you to modify, some should be left alone. You can tell which should be modified by consulting the list of files to be submitted. Anything that you aren't going to turn in *must* be left unchanged.

Other files that you may find in the assignment directories include:

**makefile** This is a standard Unix tool for project management. If an assignment comes with a `makefile`, then you can compile the program by giving the command `make` and you can clean up your directory when the project is done by giving the command `make clean`

**make.dep, *.d** These files support the `makefile`. You can ignore them. If you get a warning message about a "missing make.dep", ignore it. The `make.dep` file will be created automatically.

**executable program** In many cases, I will provide you with a compiled version of my solution. You can use this as an aid to testing your program by running your version and mine on the same inputs and comparing the outputs to see if they agree.

*Important:* Executable files can only be run on the same CPU model and operating system for which they were compiled. I typically provide, in the `bin` directory, one executable for Windows PCs and one for our Linux servers.

***assignmentName*.jar** This is the compiled version of my solution for Java programs. Typically, the program can be run via the command

```
java -jar assignmentName.jar parameters
```

where the *parameters* are any command line parameters required by the program itself. If the program requires no command line parameters, the the program can probably be executed on a Windows or other GUI-based system by double-clicking the `.jar` file in a file explorer display.

***assignmentName*.zip** Because Java often requires source code files to be arranged in subdirectories of your project, some assignments may provide source code files already arranged in the appropriate file structure. In that case, the source will be provided as a `.zip` file, that you can copy and then unpack into your own working directory.In many cases,

## 1.3 Submission guidelines

- Pay attention to the list of files you are expected to submit. Under no circumstances should you ever turn in compiled object code (.o or .obj) or executables (.exe).

  The one thing that you may turn in, in addition to the files explicitly listed in the assignment's submission list, is an optional file named README.txt, in which you may include any special notes to the grader that you feel may be necessary.

- Be very sure you have the correct names on all files. In particular, note that Unix file names are case-sensitive: "foo.cpp" and "Foo.cpp" are not equivalent, as they are on some Windows file systems. If the names are wrong, your program will not compile. If it doesn't compile on the Unix system, it's wrong. "But I had it working on my home PC!" is not an acceptable excuse.

- Do not change any other files provided with the assignment (except for your own temporary testing/debugging purposes). In particular, beware of changing interfaces! The instructor's test drivers may or may not be identical to the code provided with the assignment, but it will certainly assume that you have not altered the function names, parameter lists, or other visible interfaces. In the end, the files that you submit must compile with the *original* versions of the other files provided with the assignment.

- Pay close attention to the expected format of the output. Output that is printed in a different order, in an incorrect format, has extra or missing characters, or in which you've inserted extra debugging information, is *wrong*.

  If this seems a bit harsh, consider that, in the real world, programs must talk to each other, and to do that they must agree on I/O formats. Using an incorrect output format in those circumstances

will not only break the communications path between the programs, but will earn you the enmity of your fellow programmers because management may suspect that it was *their* input code that failed rather than your output code.

### 1.4 After the assignment is over

Assignments for this course are supposed to be a learning experience. If you are able to complete the assignment successfully, then you have presumably learned what I intended from the experience of working out your solution.

But if you were unsuccessful or only partially successful at the assignment, it's still important that you try to learn from it. The first step in doing so is to return to the assignment submission page and click on the "View Solution" button. You will see not only my solution to the assignment but also the tests that I used for the evaluation. These tests are sometimes simple input files. Other times they are shell scripts that can be executed to run the test.

Study my solution and compare it to your own. Try some of the test data listed there.

If, after doing that, you feel that you understand what was involved, then you are ready to move on. If not, it's time to ask questions.

*I'm more than happy to take questions, but if I don't hear from you, I will assume that you viewed the solution and understood it.*

## 2 Non-Programming Assignments

When asked to write essays or reports or to answer questions, you should prepare a document with your answer, using your favorite text editor or word processor, and then use the "Submit" button on the assignment page to submit that file(s). Normally, each assignment will call for a single document.

Note that, in this course, the non-programming assignments generally revolve around preparing documentation of models, designs, etc. Documentation is all about *communication*. Anything that gets in the way of communication, including careless organization, missing information, unreadable graphics, or non-portable document formats, is therefore likely to be penalized.

Unless otherwise instructed, your document must be in one of the following formats:

- plain ASCII text, saved as a .txt file.[1]
- Adobe Portable Document Format (PDF), saved as a .pdf file.

  You can produce these if your system has the Adobe Distiller software installed. Some word processors (notable the free OpenOffice suite) allow you to generate PDF directly.

[1] In some instances, this will not be acceptable. For example, if you are told to prepare a report that is to include graphs or other images, you obviously cannot submit in plain ASCII text.

You can also produce PDF from any Windows program if you have installed PDF Creator or a similar program that adds a "pdf file printer" to your list of available printers. See the course Library page for more details.

Do not assume that I will accept other document formats unless you have checked with me first! You will lose 10% of the assignment value for submitting in an unapproved format, 20% if you force me to switch to a different machine that has software capable of reading your document, and 100% if I cannot find such a machine.

## 2.1 Graphics and Drawings

In assignments that ask you to prepare charts or diagrams, you should use the appropriate drawing tools (you will find a list of some recommended ones on the Library page) to prepare your graphics.

These should then be copied and pasted into a word processing document, submitted in one of the approved formats listed earlier. Do *not* try to submit multiple separate graphics files.

The document should clearly indicate problem numbers and other identifying information associated with the assignment. If this is a team assignment, make sure that all team members' names are included.

An important part of working with graphics is the distinction between "raster" and "vector" graphics formats.

- Raster graphics formats are oriented towards producing colored pixels and subtly shaded areas, useful in photographs and artistic drawings. When images in these formats are resized or even just viewed on a device with a different resolution than the one in which it was drawn, the colors of adjacent pixels may be mixed (aliased) without noticeable degradation.

  Most of the diagrams that you are likely to produce in this class are line drawings mixed with text. Raster formats do not work well with these. When these are resized, the same mixing process results in fuzzy lines or may cause lines to disappear entirely.

  Examples of raster formats include `bmp`, `gif`, `jpeg`, and `png`. Use these with photograph-like images, not with line drawings. (If you absolutely must use one of these for line drawings, `jpeg` is probably the worst choice for this purpose.)

- Vector graphics formats are oriented towards the drawing of lines and geometric figures (including text). When images in these formats are resized, the lines are stretched by simply moving the coordinates of the end-points. Vector formats work very well with line drawings, remaining sharp and clear no matter how many times they are resized.

  They can be awkward with photographs or other images requiring subtle shading, but most vector formats will allow you to embed "boxes" of a common raster formats within a larger vector-format drawing.

  Common vector formats include `pdf`, `ps`, `eps`, `wmf`, and `emf`. Use these with line drawings.

It can be a bit harder to find graphics programs that produce vector formats, and some word processors will let you paste vector images into a document but will not display it to you until you print.

A tip: if you have a Windows drawing program that lets you move lines and shapes after placing them on the screen, it probably is using a vector format (`wmf` or `emf`) internally. In that case, if you copy-and-paste directly from your drawing program into your word processor, the graphic is probably being copied as a .wmf or .emf image even if the program does not allow you to save in those formats directly to a file.

## 3 Appendix: Protecting Your Files on the Unix System

The basic protection commands for Unix are covered in CS252. Here, however, I outline a simple setup that will make sure your files are protected from snooping without you're needing to worry about it every time you start a new assignment.

For the same of example, let's suppose that you are working on an assignment "asst1", with the instructor-provided files contained in the directory /home/cs330/Assignments/asst1.

1. If this is your first assignment for the course, create a course work area and protect it from access by anyone except yourself.

```
<b>mkdir ~/cs330
chmod 700 ~/cs330</b>
```

2. Create a working directory for this assignment:

```
<b>cd ~/cs330
mkdir asst1 cd asst1</b>
```

   You need not worry about protecting this directory, as it is inside a directory that only you can access.

3. Copy any files you need into this directory. E.g.,

```
<b>cd asst1
cp /home/cs330/Assignments/asst1/* .</b>
```

   It's important, by the way that you copy "*" and not "*.*". The latter will only copy files that have a '.' in their name. Many files do not. Most notably, the `makefile` I provide with most assignments does not.

   This **cp** command will not copy anything inside subdirectories of the assignment directory. Usually, that's OK because the only subdirectory is probably the `bin` directory where the sample solutions are kept, and you don't need a distinct copy of that.

4. You are now ready to start work.

   For most assignments, you can compile the code by just giving the command

   ```
   <b>make</b>
   ```

In future assignments, just make it a habit to do all of your assignments inside the protected `~/cs330` directory, and you won't have to worry about anyone else seeing your code.