Student Guide for 146 or

“Better, Harder, Faster, Stronger”

Written by J.J. Shepherd Ph.D.

# Introduction

This document is the serve as a guide for students in CSCE 146. It details each of the elements of the class as a way to minimize confusion. It enumerated expectations, details processes such as lab procedures, and gives several tips for how to maximize the grade in this course.

# Expectations

## Some coding experience required

This class is designed in such a way that a strong foundation in programming is necessary. The purpose is to overall make you into a better programmer with focuses on creating and using data structures, and also beginning understanding theoretical efficiency.

## Attendance

One of the surest ways to get a better grade in this course is to show up to every lecture and especially every lab. While attendance is not strictly mandatory for lectures it is strongly advised to show up. Every lecture is filled with hints and details to help with labs and homework. Unlike lectures, **lab attendance is strictly mandatory.** Even if the lab is completed, and submitted a student will receive a 0 on that lab if it is not attended.

## Work Hard and Do Not Cheat

This is considered a notoriously difficult class, but if effort is put forth a student should be able to get a passing grade. The best way to assure this will not happen is if a student is caught cheating, plagiarizing, or doing anything deemed unethical.

* 1. **Differences Between 145 and 146**

Some often consider this course to be the continuation of learning a particular programming language. While there may be some new elements of a programming language covered, the biggest focus is more on the theoretical side of computing. In addition to that here are some of the other changes between the courses.

## One Lab per Week

No longer are there two labs per week. Now labs are a little more focused on one major topic covered during the week. This means the labs are a little more difficult, and each individual lab contributes more to your final grade. It is highly recommended that you look over the labs on the website before the classes start.

## No Pair Programming

Unlike 145 each of the labs are designed to be done individually.

## Longer Assignments

There should be no surprise that in this more advanced course that assignments, especially homework, are much larger in scale. Expect to write at least a few classes for each one.

## Emphasis on Theoretical Concepts

As stated earlier, the core focus of this class is on theoretical concepts like data structures, algorithms, and efficiency. These are more deeply rooted in math, such as calculus, and the knowledge of these ideas make you an overall better programmer.

## Stricter Grading

At this stage, we expect you to have a decent grasp on basic programming. As such, each assignment will be graded stricter with an emphasis on better coding style and technique. It is not good enough to just have a program that works.

# Class Structure

## Lectures

Lectures are one hour and fifteen minute sessions that are done twice a week. The purpose of this is to introduce new topics that will be further detailed in subsequent labs and homework. Most lectures consist of a presentational component, and a coding example detailing the concept. Any of the code given in lecture may be used, and altered as a way to solve various assignments. However, the lecturer is not able to post any source code online. Bringing a laptop to follow along is not required, but it is strongly encouraged. Attendance for lectures is not (always) mandatory, but it is strongly advised that you attend all lectures. However, if attendance starts becoming noticeably low the instructor may opt to give a graded assignment (like a quiz) to encourage better numbers.

## Labs

Along with lectures there will be one lab every week. These are long sessions that are conducted by the TA’s with a focus of completing a coding assignment. **Attendance is mandatory for every lab**. A student will receive a grade of “0” if they do not attend the lab. More details can be found in INSERT SECTION.

## Supplemental Instruction (SI)

To aid in learning this material there will be weekly supplemental instruction (SI) sessions every week. These are purely option, and student led information sessions. While they SI Leader cannot directly give answers to assignments, they are poised to assist in any topics that may have been confusing and aid in concepts related to the assignments.

# Roles

There are a number of people who make sure this class conducted in an efficient fashion. Here are a few of their roles.

## Instructor

The instructor conducts lectures, creates assignments, and oversees the inner workings of the class. They coordinate with TA’s, SI Leaders, and students to ensure the class runs as smoothly as possible. They may take on some grading when needed, and have the ultimate say on any grade in the course.

## Teaching Assistants (TA’s)

Accompanying the instructor is a bevy of teaching assistants (TA’s). Their main goal is to conduct labs and grade assignments. Typically TA’s are either graduate or undergraduate students. Graduate students take the lead on labs, and are the ones who grade assignments. The undergraduate TA assists in the lab, and does not grade any works.

## Supplemental Instructor

Along with TA’s there are supplemental instructors that aid in understanding outside of the class. These undergraduate tutors host a number of sessions per week, and provide additional material to help with the understanding of the subjects and assignments.

# Labs

Labs are an essential element of this course, and contribute a sizable part of your grade. Labs are meant to reinforce concepts covered in the lectures, and help set the stage for other assignments such as homework. The following details how labs work

## Structure of the Lab

### Sign-In

Before the lab begins, each you must sign your name on a sign-in sheet. Since attendance for labs is mandatory, failure to sign in will result in a “0” for that assignment.

### Overview

The beginning of the lab starts out with an explanation the problem. The TA’s will give a short description of the lab along with its objectives.

### Short Lesson (Optional)

If a concept or a lab is on the more difficult side, the TA may conduct a short lecture going over its finer points. Careful attention should be paid to these parts when they occur as they may answer a several questions that may arise during the coding session.

### Question and Answer

Once the programming session begins you may ask your TA’s for help or advice. The TA is instructed to only spend roughly five minutes at a time working with each pair, so they can answer as many questions as possible. It is important for you to have a clear question in mind and to avoid statements like, “I don’t know what I’m doing.”

### Checking / Grading

Once you have finished your lab, call over the TA’s to have them check you off. Unless it requires more detail work, the TA will assess your code and assign a grade immediately. The TA may deduct points for a multitude of reasons, but here are some of the most common ways you can lose points:

* The program has logic, syntax, or run-time errors
  + The program won’t compile
  + Lack of error checking
  + Expected output is not the expected result
* The programs style needs improvement
  + Poorly named variables
  + Magic numbers
  + Method’s or variables scope being improperly set to private or public
* Bad commenting
  + Missing you and your partner’s name
  + Little to no comments throughout the code
  + Over commenting or pointless comments
* Vulgarities in the code

These are only a few examples of where points may be deducted. For further explanation on grading look at the “Grading” section of this guide.

### Wrap-Up

Before the lab, you need to make sure the TA has checked over your work, and you have been signed into the lab. Once you have completed the assignment, had the TA check it, and submitted it to Dropbox then you may leave the lab.

## Example Lab

Lab 10  
Insertion Sort

**Objective:**

Write a program that takes in any user defined number values and then performs insertion sort resulting in an array of values in ascending order.

* First the user must input how many values they are entering
* Next the user enters that many values, which populates an array of the size previously entered.
* Using TWO arrays then perform insertion sort
  + The first array stays unsorted
  + The second array is sorted
* Pseudo-code for insertion sort

// a is the first array

for(index = 0; index < a.lenght-1; index++)

put the value of index a[index] into the second array in the correctly sorted position

(IE the values at a lower index are less than the value. The values at a higher index are greater than the value.)

* When inserting a number in the middle of the array all other numbers ahead of it must be shifted forward
  + HINT: Start from the back and pull the values forward

5

6

8

7

4

1

5

6

8

i = 3

test = a[i] = 7

0

1

2

3

4

5

j=0

**j>=i or**

**a[i]>=b[j]?**

0>=3 False!

5>=7 False!

Move on

0

1

2

3

4

5

j=1

**j>=i or a[i]>=b[j]?**

1>=3 False!

6>=7 False!

Move on

j=2

**j>=i or a[i]>=b[j]?**

2>=3 False!

8>=7 **TRUE!**

-Move everything starting at b[j] till j>=i up one index

-Insert a[i] at b[j]

5

6

7

8

0

1

2

3

4

5

5

6

8

0

1

2

3

4

5

a

b

b

b

* Print both the sorted and unsorted arrays

**Example Dialog:**

**Welcome to the insertion sorter**

**Please enter the number of values you would like to sort**

**6**

**Please enter the number at 0**

**5**

**Please enter the number at 1**

**6**

**Please enter the number at 2**

**8**

**Please enter the number at 3**

**7**

**Please enter the number at 4**

**4**

**Please enter the number at 5**

**1**

**The unsorted array is**

**5 6 8 7 4 1**

**The sorted array is**

**1 4 5 6 7 8**

**Done!**

**Finally:**

Upload the .java file to the dropbox under Lab10

**Additional Questions for the Lab Report**

1. Could insertion sort be implemented using only one array?
2. Given this array demonstrate each step of insertion sort as described in the lab. Use two arrays.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Value | 6 | 5 | 2 | 1 | 3 |

## Other FAQ’s about Labs

Here are some frequently asked questions about labs in general.

* “May I start working on the lab early?”
  + Yes you may, and it is encouraged to at least look over the lab before it begins. Generally labs are posted a few hours before the labs begin.
* “I’ve already finished the lab, so do I still need to come in?”
  + Yes, you always need to attend lab. Due some University policies lab attendance is mandatory. If you did happen to finish it early then you may show up, sign in, show your assignment, and finally leave.

# Lab Reports

After every lab, a detailed report is due by the next lab. This is not meant to be a lengthy document, and it serves as gaining better understandings of the assignment and help serves as a way to avoid making the same mistakes repeatedly. **These are meant to be done individually**. The structure of the lab report goes as follows.

## Lab Report Format

## Problem

State the given problem clearly in one’s own words. Do not just copy and paste the description given in the lab.

## Proposed Solution

Give a hypothesized algorithm to solve the problem. This description must be a detailed and high-level without using implementation details (in other words no formal code). One way to think of it is it combines both the hypothesis and the procedure. Flow Charts and graphics are strongly encouraged.

## Tests and Results

Show a sufficient number of tests with the results demonstrating that the proposed solution works, which includes boundary conditions. Also show that the program works or halts properly for invalid values.

## Problems Encountered

Enumerate the issues that arose from creating this solution. Include major syntax, run-time, and logical errors with their respective solutions.

## Conclusions and Discussion

Sum up the lab and the results. Also discuss other ways to have solved the problem in a better way with supporting evidence.

## Additional Questions

There may be additional questions that will be provided in order to demonstrate the understanding of the subject.

## Example Lab Report

Lab 10 Report: Insertion Sort

# Problem

We had to write a program in which a user populated an array of integers and then it was sorted using insertion sort. Finally, the program printed out the sorted array to the console.

# Proposed Solution

1. Prompt the user for the size of the array
2. If the size is a negative value then quit the program
3. Create an array of integers of size given in step 1
4. For each element in the array
   1. Prompt the user to input a value
   2. Store that value as that element of the array
5. Create a second array the same size as the first
6. For each element in the first array
   1. For each element in the second array
      1. If we are at the end of the second array then insert that element
      2. Otherwise if we find a value in the second array that is smaller than the examined value in the first
         1. Shift the values in the second array right
         2. Insert the value of the first array into the second
7. For each element in the second array print the values thus in printed order

## Flow Chart

Prompt the user for the size of the array

Size <- user Input

Size < 0

No

Program Ends

Create unsorted array “us” of size Size

Yes

i<-0

i < us.length

Prompt user for a value

us[i] <- user input

i <- i+1

Yes

Create a second array “s” of size Size

i<-0

i<us.length

j<-0

j<us.length

No

Yes

At the end of the sorted array so

i = j

s[j] <- us[i]

Yes

An element is out of order thus

us[i] < s[j]

No

Yes

s[j] <- us[i]

Shift all elements start at j to the right by 1

j <- j+1

i<- i+1

No

Print all elements of the sorted array “s”

No

# Tests and Results

|  |  |  |
| --- | --- | --- |
| Test Case | Test Input | Result |
| A reversed order array | 7 6 5 4 3 2 1 | 1 2 3 4 5 6 7 |
| An in-order array | 1 2 3 4 5 6 7 | 1 2 3 4 5 6 7 |
| Random order array | 2 4 3 1 6 5 7 | 1 2 3 4 5 6 7 |
| A larger random order array | 10 7 5 6 4 2 3 1 9 8 | 1 2 3 4 5 6 7 8 9 10 |
| Negative size was entered | -1 for the size | The program quits |
|  |  |  |

# Problems Encountered

Using the incorrect index for unsorted array and the sorted array was a major problem. It was difficult to keep track of one to the other until I went back and renamed the arrays and the variables using a more descriptive but much longer identifier.

Index out of bounds exceptions came up as I accidentally was using <= the array’s length for loop’s Boolean expression instead of <. I’ve made a note that the last valid index in an array is the length-1.

Another index out of bound exception arose when I forgot to check to make sure the entered size of the array was non-negative. That was fixed by putting an if-statement that halted the program when that occurred.

# Conclusions and Discussion

In this lab we sorted an array using the insertion sort algorithm. The way it worked was creating two arrays and then inserting the elements of the first array in the correct order in the second. Shifting required finding where the value belonged and shifting values over.

While this algorithm works, I think a better solution for sorting is sticking to bubble sort. First, bubble sort is much simpler to code as there isn’t a need for shifting values. One simply swaps values until the correct location is found. Second bubble sort only requires one array. There is a version of insertion sort that uses one array, but its implementation seems to be a little trickier. I see the benefit of using insertion sort in some cases, but for small cases like this I would prefer to use bubble sort.

# Additional Questions

1. Could insertion sort be implemented using only one array?

Yes I believe it can. For instance if it were to examine a particular index, and then examine each element before to determine if it needs to be inserted elsewhere or remain where it is.

1. Given this array demonstrate each step of insertion sort as described in the lab. Use two arrays.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Value | 6 | 5 | 2 | 1 | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Unsorted | 6 | 5 | 2 | 1 | 3 |
| Sorted | - | - | - | - | - |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Unsorted | 6 | 5 | 2 | 1 | 3 |
| Sorted | 6 | - | - | - | - |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Unsorted | 6 | 5 | 2 | 1 | 3 |
| Sorted | 5 | 6 | - | - | - |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Unsorted | 6 | 5 | 2 | 1 | 3 |
| Sorted | 2 | 5 | 6 | - | - |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Unsorted | 6 | 5 | 2 | 1 | 3 |
| Sorted | 1 | 2 | 5 | 6 | - |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 |
| Unsorted | 6 | 5 | 2 | 1 | 3 |
| Sorted | 1 | 2 | 3 | 5 | 6 |

# Homework

Homework mostly consists of programming assignments that are longer than a usual lab. These are posted early in the week, and are commonly due at the end of the week. Homework’s subject deals with topics covered previously in lecture, and are used to expand the overall concepts. These are graded in a similar way as the labs, but are graded more strictly. Also unlike labs homework are meant to be done **individually** and not in pairs or groups.

# Exams

Half of your grade is based on a variety of exams. Written and lab exams always occur in the same week. Make sure to check the syllabus for those times.

## Written (In Class) Exams

Written exams occur twice in the semester, and are administered during the lecture time. These are comprehensive exams that are **closed book, notes, and electronic devices**. Any unauthorized use of these materials will result in an automatic zero. They may have a number of question types such as vocabulary, short answer, and small programs. Before each exam, a study guide will be provided.

## Lab Exams

Lab exams are generally the first lab during the week of a written exam. These work a little differently than traditional labs. These will not be posted online, so the TA will provide the assignment at the beginning of the lab. Also unlike written exams, these are **open book and notes**. However, **it is NOT open internet**. Besides the Dropbox website and the instructor’s website, you may not use any other online resources. A violation of this rule will result in an automatic zero, and academic integrity will be contacted.

## Final (Written) Exam

The final exam will be a comprehensive exam of the entire course. By university policy, you have 2 ½ hours to complete this exam. Just like the other written exams this is **closed books, notes, and electronic devices**. However, the final holds a more weight than the other written exams. **If you cannot make score a sufficiently passing grade on the final exam then you will make an “F” in the overall course.** This also means the final exam is not exemptible. Also there is no final lab exam.

# Extra Credit

Generally, after the first exam a series of extra credit problems will be posted. These are due at the end of the semester, and consist of large, intricate projects. Some elements in the extra credit work are not and will not be discussed in the class. The purpose is to allow you the ability to go and search for materials that will augment the overall class, and provide a second chance to earn some points on concepts that you may have not fully grasped at an earlier time. Each one of these assignments will add a number of points to your grade.

# Grading

## Programming Assignments

Programming assignments, including labs and homework, are graded based on the following criteria.

## Correctness

This determines if the program works and contains a number of errors. Some of the most common errors are syntax, run-time, and logic errors. Any of these errors can result in points being deducted from the assignment’s score, and the amount depends on the severity of the error.

Syntax errors may be misspelling a variable’s name, forgetting to put a semi-colon at the end of a statement, or calling a method with incorrect parameters. When these exist the program cannot compile, and therefore cannot run. These are generally easy to fix and most of the time IDE’s will indicate these types of errors with a red line underneath the statement.

By contrast, run-time errors happen while the program is executed. These lack syntax errors, and as such the program compiles fine. However, during the program’s execution it crashes for a number of reasons. Some of these reasons include calling methods from a null object (null reference exception), indexing outside the bounds of an array, or parsing a string into a numeric value when none exists. These can generally be fixed by ensuring objects are constructed, and coding in error checking conditions.

Finally, the logic error is considered by many to be the worst kind. It lacks syntax errors and for the most part run-time errors. However, the intended output was not achieved from the given input. This type of errors can be caused by any number of reasons. Some common ones would be the incorrect use of inclusive or exclusive inequality signs (IE < instead of <= and vice versa). Others could be an incorrectly constructed Boolean expression.

## Style

Simply having the program work is not always enough. There are standard practices in programming that make code comprehensible and reusable. Points may be deducted for the following, but not limited to, these reasons.

* Bad Formatting: Proper indentations and use of curly braces (“{}”) are key for reading code. When code does not follow these rules it can become a jumbled, impossible to read mess.
* Poor Naming Conventions: When it comes to variables, constants, class names, and methods there are a number of conventions to how we name them. First all of these should have names that make sense within the program and not something arbitrary or vulgar. Second if it is a variable, class name, or a method it should be “Camel Cased”, and constants should be all caps with underscores (“\_”) separating out each word.
* Magic Numbers: These are numbers that are arbitrarily placed in code without any explanation. The best way to avoid these is by creating constants which will name that number.
* Bad Scope: Making variables and methods either public or private in the appropriate times is crucial to having safe code. Inappropriate use of this can create potential bugs in programs.
* Improper use of Statics: Methods and variables work better in some cases when they are static in memory. Not doing this properly can create inefficient programs. This also can relate to the overuse of global values which is also discouraged.

## Commenting

Related to style, commenting is key to making code readable and maintainable. Points may be deducted if there are too little comments or too many comments. First you must always have your name at the top of the program. However, having just your name is not always enough, and then again having comments after every single statement can make the program unreadable. The general rule of thumb is a comment should describe what the statements are doing up until the next comment. Also if you perceive some code as being confusing or hard to follow if someone else to read your program, then that is definitely a good spot for a comment.

## Overall Grading

According to the syllabus you can calculate your final grade by the following equation:

This does add up to 110%, but the grade is only calculated out of 100%, and this class follows the college grading scale where an “A” is 90 and above, “B+” is 85 to 89, “B” is 80-84, etc. Though, as stated in a previous section**, if you cannot make a sufficiently passing grade on the final exam, then you will receive an “F” in the course**.

# Cheating

Cheating is defined in this class as getting unauthorized assistance from an outside source. This could be copying code you found on the internet, taking someone else’s code that was given to you, or stealing another student’s code. The best way you can consider any action as cheating is if it the code or work did not come from your mind, through your keyboard, into your computer. Assignments are checked extensively for acts of plagiarism. The first offense will result in the grade of 0 for the assignment, and the second offense will result in the grade of “F” for the class and academic integrity will be notified.

**DO NOT CHEAT IN THIS COURSE!** The purpose of this course is to learn and hone a skill. Cheating will not help you get better, and hurts everyone else who is trying. There are plenty of available resources for this class that can help you.

# Helpful Tips and Resources

## Setting up Development Software / IDE

An integrated development environment (IDE) makes the difference between having to spend days on software versus a few hours. These applications aid in the development of software and can be easily installed on your personal systems. We recommend the following software for this course:

* Dr. Java

This is great for beginners and simple to install. Here is the link with installation instructions <http://www.cs.cornell.edu/courses/cs1110/2008fa/handouts/handout3.html>

* Eclipse

This is a common Java IDE that is used in many classes. Here are the instructions for installing Eclipse <https://wiki.eclipse.org/Eclipse/Installation>

* IntelliJ

This very popular Java IDE comes bundled with many features and allows developers to quickly port to platforms like Android. Here is the installation link <https://www.jetbrains.com/help/idea/install-and-set-up-intellij-idea.html>

* NetBeans

This good yet large IDE is great for developing graphic user interfaces (GUI). The installation instructions are found here <https://netbeans.org/community/index.html>

## Debugging

Finding the errors in your assignments is the most difficult aspect to this class. Here are some common strategies and tips to eliminate bugs.

## Common Errors and Solutions

There are a number of common errors that arise while developing code. First and foremost look at the line number of the error. This is commonly found in the console’s print out, and help direct you to the source of the problem (but not always). Now here are a few with some solutions to aid you further.

* Forgotten Semicolon

Semicolons appear at the end of most statements. Forgetting one will yield a syntax error, and the IDE may prompt you to place one at the end of the statement.

* Incorrectly named variables, methods, or classes (“Cannot be resolved”)

Variables, methods, and classes must be called using their EXACT name. This means it has to be spelled the exact same way and the case matters. For instance if one were to create a variable name “taco”, and then try to access it by calling “Taco” this would result in this error, because the “T’s” do no match. To resolve these errors check the spelling and the case exactly.

* Mismatching curly braces “{}” or parenthesis “()”

For every curly brace and parenthesis they must have an open one (“(“ and “{“) and a closed one (“)” and “}”). This syntax error can sometimes be difficult to locate especially if the code is not well formatted. The key to avoiding this error is to make sure your code is formatted neatly, and to simply check every single instance or either to make sure it matches in the correct place. Avoid arbitrarily putting closing curly braces at the end of the code as that can cause many logic errors to occur.

* Putting a Semicolon after an “If”, “While”, or “For” statement

Nearly every other statement requires a semicolon at the end of it, but in the case of “if”, “while”, or “for” you DO NOT put a semicolon after the closing parenthesis. If you do this the statement will not work, and you won’t receive a syntax error.

* The “=” is different from “==”

The single equals (“=”) is an assignment operator which takes the value from the right hand side and stores it in the variable on the left hand side. The double equals (“==”) is a comparison operator that yields a true or false value if the left hand side is equal to the right hand side. These type of errors frequently occur inside of “if” or “while” statements. Make sure you’re using the correct instance or logic errors will happen.

* Index Out Of Bounds Exception

This run-time error happens when the code tries to access any values outside the bounds of an array. The only valid indices in arrays are from 0 to the array’s length minus one. Make sure if you’re using loops that the inequality does not accidentally exceed the bounds – either being equal to the array’s length or access an index that has a negative value.

* Infinite Loops

If there is no way a loop, such as “while”, “do-while”, and “for”, can ever end the program will be stuck in an infinite loop. Check to make sure your Boolean expression for these are correct, and there is a possible stopping condition.

* Null Reference Exception

This can be caused by any number of reasons, but it all depends on if you are calling a method or trying to access a value from an unconstructed object. Make sure you have either constructed the object, or have not set that object to “null” for some reason.

* Infinite Recursion

Similar to infinite loops, this happens when there is not a halting condition when recursively calling methods. This is usually indicated with a run-time error saying “you’ve exceeded the allowed main memory”. Just like infinite loops make sure there is a reachable halting condition in your recursive calls.

## Print Statements

In the case of logic errors one sure way to check to see what’s going on is using “Print Statement”. In Java, using “System.out.println()” can be an extremely effective way of seeing the inner workings of a program. Printing out the values of variables can indicate if they are being set correctly. Placing print statements inside of methods and inside the body of an “if”, “while”, etc. can check if everything is working the way you intended. Once the problem is solved you need to remove those print statements. Also avoid using vulgarities in print statements.

## Debugging Tools

For larger and more intricate projects, the use of debugging tools built into an IDE (such as Eclipse, IntelliJ, and NetBeans) may be crucial. These tools give insight into the program’s “call stack”, values of variables, and allow the programmer to step through each line of code. For more information, read up on your preferred IDE’s debugging tools.

## Books and Online Resources

Outside resources can be essential to becoming a great computer scientist, or at least passing this course. Here are some recommended external materials.

## Your Text Book

The text book recommended for the course can be a vital resource. Unfortunately, there is not enough time to go over every element of a subject. The book provides the extra pieces that are either glossed over or left out of the course for the sake of time.

## Book “How to Think Like a Programmer”

The most difficult part of coding is being able to approach problems in a logical and pragmatic way. If you are having some difficulty translating your ideas into working code then the book “Think Like a Programmer: An Introduction to Creative Problem Solving” by V. Spraul is a great place to start. This book gives several examples and step-by-step instructions as to how to break down and solve a problem. However, the examples are done in C++ and not in Java.

## Video Tutorials

Video sites, like YouTube, are not only a great source for makeup tutorials, but also great for understanding harder concepts.

## Online Practice

Here are a few online resources for extra help and extra problems:

* <https://www.hackerrank.com/>
* <https://code.google.com/codejam/>
* <https://coderbyte.com/>
* <https://www.codeeval.com/>

# Other Frequently Asked Questions

## May I make-up <Insert Assignment Here>?

Only in extreme cases can any student make up any assignment after the fact. Each of these has to be accompanied with some sort of proof of the absence. A rule of thumb is if you were not sick enough to go to a doctor then you should attend class.

However, if beforehand you know you are going to miss the deadline for an assignment because of something important, then arrangements can be made for a student to turn in the assignments at a later date. A student must inform the instructor at least a week before the absence, so that the proper arrangements can be made.

## Is it possible to get partial credit for late assignments?

Much like make-up work, this is only done in extreme cases, and requires an excuse.

## I cannot make it to your office hours, but is there a way I can meet with you?

Office hours are set up specifically, so students can interact with the instructor one-on-one. However, outside of those hours are filled with other aspects of the job such as lesson planning, assignment creation, meetings, grading, etc. It is possible to meet during those outside times, but schedules fill up fast. Best way is to plan early, and be patient.

## Can you put your examples online?

No. Best way to get the examples is to show up to lecture, and follow along.

## I cannot attend an exam. Is there any way I can take it another time?

Any time an exam has to be rescheduled it must be for a valid reason. Similar to make-up assignments, this excuse has to have some sort of proof associated with it. If it’s valid then a time can be negotiated to make-up an exam. Also for the final exam it should be completed BEFORE the final exam’s date.

## I would like a Letter of Recommendation

Letter of recommendations are given to top students. If you would like an instructor to write one then you must do the following:

* Request one early in the semester
* Send the link to recommendation submission
* Send a resume
* Send a CV (Curriculum Vitae)

# Contact

## Office Hours

This is the surest way you can communicate to the instructor. Every week, the instructor holds a number of hours reserved to have one-on-one interactions with students. You may feel free to stop by and ask questions regarding assignments, grades, or any type of consultation. Meeting with the instructor outside of this time may be difficult due to their schedule, and if this is the case then let the instructor know as soon as possible.

## Emailing the Instructor

Emailing the instructor is the next best way, but since they have a large number of students to work with, it may take a while to respond. Also for this reason It is hard to keep track of all students, so the best way to get a faster turnaround is to include your name, your class, and your section number.

## Emailing TA’s and SI Leaders

Besides the instructor you may need to contact your TA’s or SI Leaders. You can find their email addresses on either the directory on sc.edu, Dropbox, or Blackboard.

# Disclaimer

**\*\*\*Everything in this document is subject to change\*\*\***