CS 2110 Homework 6
Intro to Assembly

Sam, Colin, Elton, Matt, Nimish

Spring 2020

Contents

1 Overview **3**
- 1.1 Purpose . . . 3
- 1.2 Task . . . 3
- 1.3 Criteria . . . 3

2 Detailed Instructions **4**
- 2.1 Part 1 . . . 4
- 2.2 Recursive Multiplication . . . 4
  - 2.2.1 Pseudocode . . . 4
- 2.3 Part 2 . . . 4
- 2.4 Recursive GCD . . . 4
  - 2.4.1 Pseudocode . . . 4
- 2.5 Part 3 . . . 5
- 2.6 Replace values in Linked List . . . 5
- 2.7 Linked List Data Structure . . . 5
  - 2.7.1 Pseudocode . . . 5
- 2.8 Part 4 . . . 6
- 2.9 Map . . . 6
  - 2.9.1 Pseudocode . . . 6

3 Checker **6**

4 Deliverables **7**

5 Appendix **8**
- 5.1 Appendix A: LC-3 Instruction Set Architecture . . . 8
- 5.2 Appendix B: Running Autograder and Debugging LC-3 Assembly . . . 9
- 5.3 Appendix C: LC-3 Assembly Programming Requirements and Tips . . . 12

5.4 Appendix D: Rules and Regulations 13
5.4.1 General Rules 13
5.4.2 Submission Conventions 13
5.4.3 Submission Guidelines 13
5.4.4 Syllabus Excerpt on Academic Misconduct 14
5.4.5 Is collaboration allowed? 14

1 Overview

1.1 Purpose

Now that you've been introduced to assembly, think back to some high level languages you know such as Python or Java. When writing code in Python or Java, you typically use functions or methods. Functions and methods are called subroutines in assembly language.

In assembly language, how do we handle jumping around to different parts of memory to execute code from functions or methods? How do we remember where in memory the current function was called from (where to return to)? How do we pass arguments to the subroutine, and then pass the return value back to the caller?

The goal of this assignment is to introduce you to the Stack and the Calling Convention in LC-3 Assembly. This will be accomplished by writing your own subroutines, calling subroutines, and even creating subroutines that call themselves (recursion). By the end of this assignment, you should have a strong understanding of the LC-3 Calling Convention and the Stack Frame, and how subroutines are implemented in assembly language.

1.2 Task

You will implement each of the four subroutines (functions) listed below in LC-3 assembly language. Please see the detailed instructions for each subroutine on the following pages. We have provided pseudocode for each of the subroutines, and suggest that you follow these algorithms when writing your assembly code. Your subroutines must adhere to the LC-3 calling convention.

1. `mult.asm`
2. `gcd.asm`
3. `ll.asm`
4. `map.asm`

More information on the LC-3 Calling Convention can be found on Canvas in the Lab Guides and in Lecture Slides L12 and L13. More detailed information on each LC-3 instruction can be found in the Patt/Patel book Appendix A (also on Canvas under LC3 Resources). Please check the rest of this document for some advice on debugging your assembly code, as well some general tips for successfully writing assembly code.

1.3 Criteria

Your assignment will be graded based on your ability to correctly translate the given pseudocode for subroutines (functions) into LC-3 assembly code, following the LC-3 calling convention. Please use the LC-3 instruction set when writing these programs. Check the deliverables section for deadlines and other related information.

You must obtain the correct values for each function. In addition, registers R0-R5 and R7 must be restored from the perspective of the caller, so they contain the same values after the caller's JSR subroutine call. Your subroutine must return to the correct point in the caller's code, and the caller must find the return value on the stack where it is expected to be. If you follow the LC-3 calling convention correctly, each of these things will happen automatically.

While we will give partial credit where we can, your code must assemble with no warnings or errors. (Complx will tell you if there are any.) If your code does not assemble, we will not be able to grade that file and you will not receive any points. Each function is in a separate file, so you will not lose all points if one function does not assemble. Good luck and have fun!

2 Detailed Instructions

2.1 Part 1

2.2 Recursive Multiplication

For the first part of this homework, we will be writing a function that computes the multiplication of two numbers, a and b, recursively. Note that only one of a or b will be negative.

2.2.1 Pseudocode

You may remember your implementation for iterative multiplication; this wont be much different. We will be taking in two parameters a and b, and returning $a * b$. Only one of a or b will be negative in any given call. The pseudocode is as follows:

```
mult(int a, int b) {
    if (a == 0 || b == 0) {
        return 0;
    }

    // Since one number might be negative, we will only decrement the larger number.
    // This ensures that one parameter will eventually become zero.
    if (a > b) {
        var result = b + mult(a - 1, b);
        return result;
    } else {
        var result = a + mult(a, b - 1);
        return result;
    }
}
```

2.3 Part 2

2.4 Recursive GCD

For this part of the assignment, you will be recursively determining the greatest common divisor of two integers passed in as arguments. The greatest common divisor d of two integers a and b is defined as the greatest integer that divides both a and b. If you would like additional reading on how Euclid's Greatest Common Divisor Algorithm works, please read https://crypto.stanford.edu/pbc/notes/numbertheory/euclid.html for a basic rundown.

2.4.1 Pseudocode

Following is the pseudocode for determining the GCD of two integers a and b.

```
gcd(int a, int b) {
    if (a == b) {
        return a;
    } else if (a < b) {
        return gcd(b, a);
```

```
        } else {
            return gcd(a - b, b);
        }
    }
```

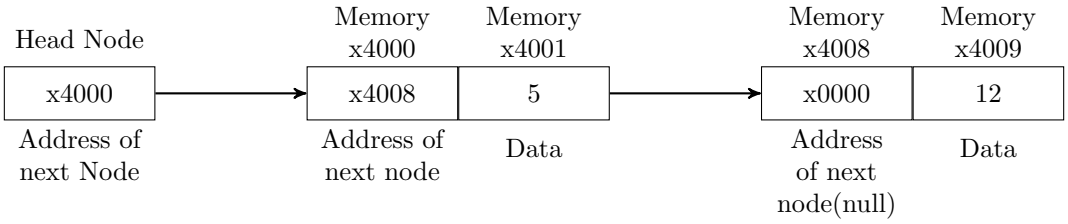
2.5 Part 3

2.6 Replace values in Linked List

For this part of the assignment, you will be writing a recursive function that for each node i of a linked list L will check if the node's data $d_i == r$ for some integer r. If $d_i == r$, you will set $d_i = 0$. If $d_i \neq r$, you will set $d_i = d_i * r$. Then, you will return the current node. The parameters for the function will be the address of the head of the linked list L, and the integer r.

2.7 Linked List Data Structure

The below figure depicts how each node in our linked list is laid out. Each node will have two attributes: the next node, and a value for that node.



2.7.1 Pseudocode

The pseudocode for this function is as follows. **Note: null has a value of 0.** Since memory address zero is used to hold part of the trap vector, we can use zero as a value distinguished from all of the memory addresses in our linked list.

```
replaceValueLL(Node head, int r) {
     if (head == null) {
         return head
     }
     if (head.data == r) {
         head.data = 0
     } else {
         head.data = head.data * r
     }
     replaceValueLL(head.next, r)
     return head;
}
```

2.8 Part 4

2.9 Map

For this part of the assignment, you will be implementing a map function. The parameters of the function are as follows: the address of an array A, the length of array A, and the address of function named $func$. For each value $A[i]$ in array A, you will perform the following operation:

$a[i] = func(a[i])$

Then, you will return the address of the modified array A.You may have seen a similar function to this in java. If you'd like to refresh your memory, please visit https://codingbat.com/doc/java-functional-mapping.html for a brief overview.

Note: unlike previous questions, you can't use JSR to jump to the provided function. JSR takes in the PC-relative location of a subroutine as one of its operands, but some of the provided functions are located too far away for JSR to reach. What function similar to JSR can be used instead?

2.9.1 Pseudocode

Following is the pseudocode for map.

```
map(Array A, int len, function func) {
    for(i = 0; i < len; i++) {
        A[i] = func(A[i]);
    }
    return A;
}
```

3 Checker

To run the autograder locally, follow the steps below depending upon your operating system:

- Mac/Linux Users:
  1. Navigate to the directory your homework is in. **In your terminal, not in your browser**
  2. Run the command `sudo chmod +x grade.sh`
  3. Now run `./grade.sh`
- Windows Users:
  1. On **docker quickstart**, navigate to the directory your homework is in
  2. Run `./grade.sh`

Note: The checker may not reflect your actual grade on this assignment. We reserve the right to update the checker as we see fit when grading.

4 Deliverables

Turn in the files

1. `mult.asm`
2. `gcd.asm`
3. `ll.asm`
4. `map.asm`

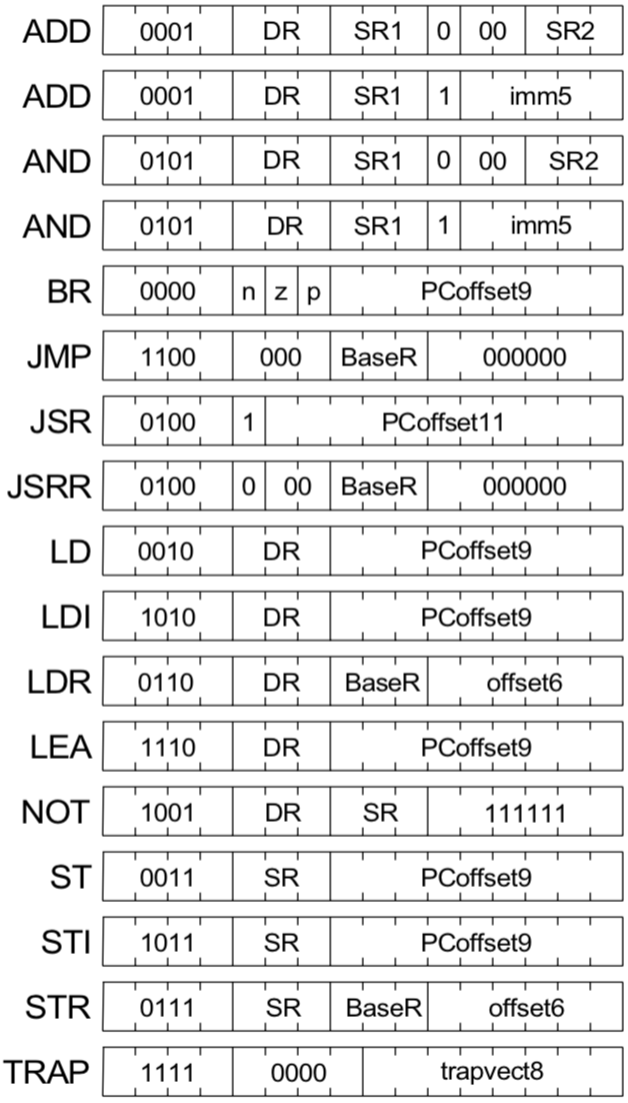
on Gradescope by Thursday, March 5th at 11:55pm.

There will be a 6 hour late period until March 6th at 5:55am. You will receive a 25% deduction during this late period. Anything after that will result in an automatic 0.

Note: Please do not wait until the last minute to run/test your homework, history has proved that last minute turn-ins will result in long queue times for grading on Gradescope. You have been warned.

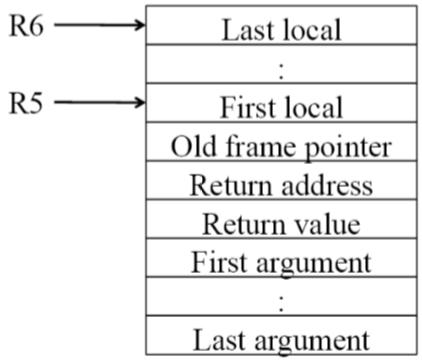
5 Appendix

5.1 Appendix A: LC-3 Instruction Set Architecture



Trap Vector	*Assembler Name*
x20	GETC
x21	OUT
x22	PUTS
x23	IN
x25	HALT

Device Register	*Address*
Keybd Status Reg	xFE00
Keybd Data Reg	xFE02
Display Status Reg	xFE04
Display Data Reg	xFE06



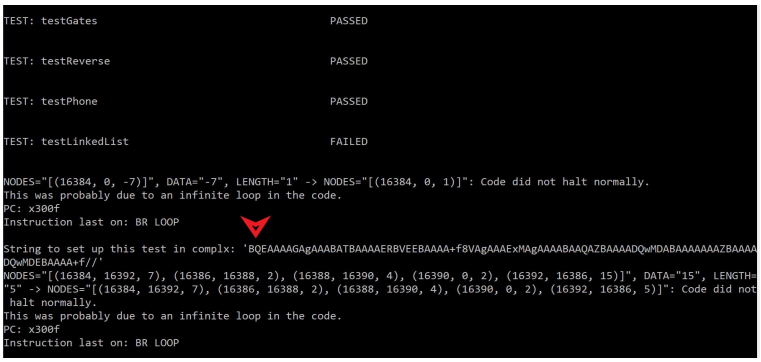
5.2 Appendix B: Running Autograder and Debugging LC-3 Assembly

When you turn in your files on gradescope for the first time, you might not receive a perfect score. Does this mean you change one line and spam gradescope until you get a 100? No! You can use a handy tool known as tester strings.

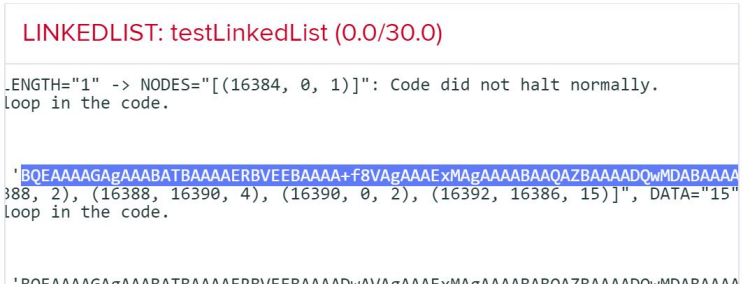
1. First off, we can get these tester strings in two places: the local grader or off of gradescope. To run the local grader:

   - Mac/Linux Users:
     (a) Navigate to the directory your homework is in. **In your terminal, not in your browser**
     (b) Run the command `sudo chmod +x grade.sh`
     (c) Now run `./grade.sh`
   - Windows Users:
     (a) On **docker quickstart**, navigate to the directory your homework is in
     (b) Run `./grade.sh`

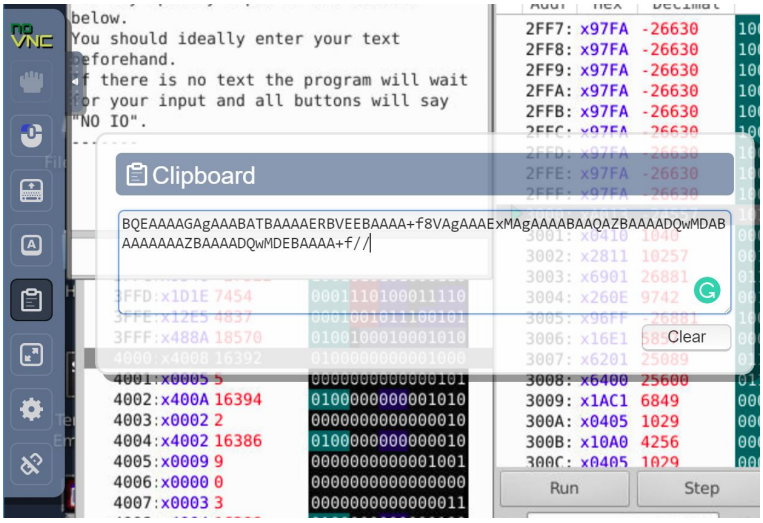
   When you run the script, you should see an output like this:



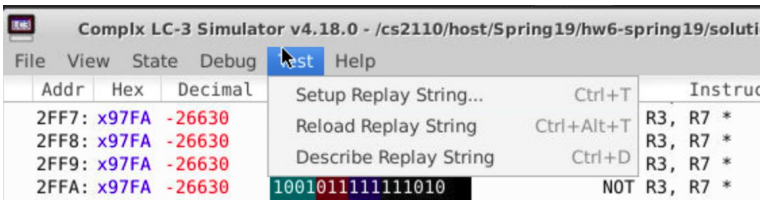
   Copy the string, starting with the leading 'B' and ending with the final backslash. Do not include the quotation marks.
   Side Note: If you do not have docker installed, you can still use the tester strings to debug your assembly code. In your gradescope error output, you will see a tester string. When copying, make sure you copy from the first letter to the final backslace and again, don't copy the quotations.



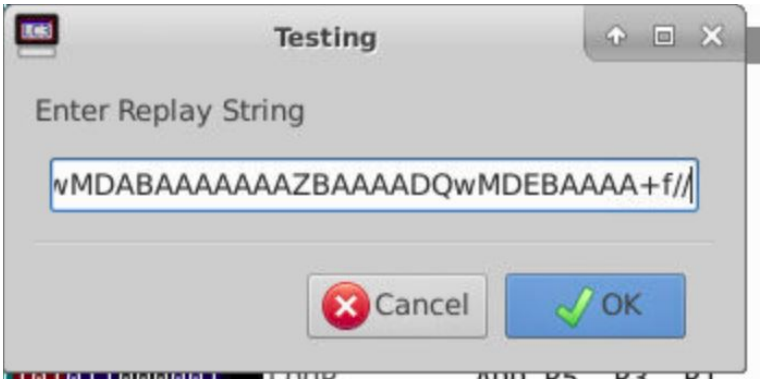
2. Secondly, navigate to the clipboard in your docker image and paste in the string.



3. Next, go to the Test Tab and click Setup Replay String

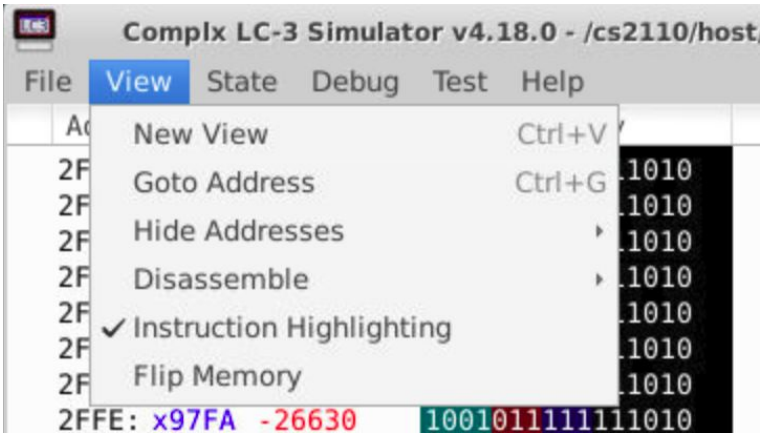


4. Now, paste your tester string in the box!

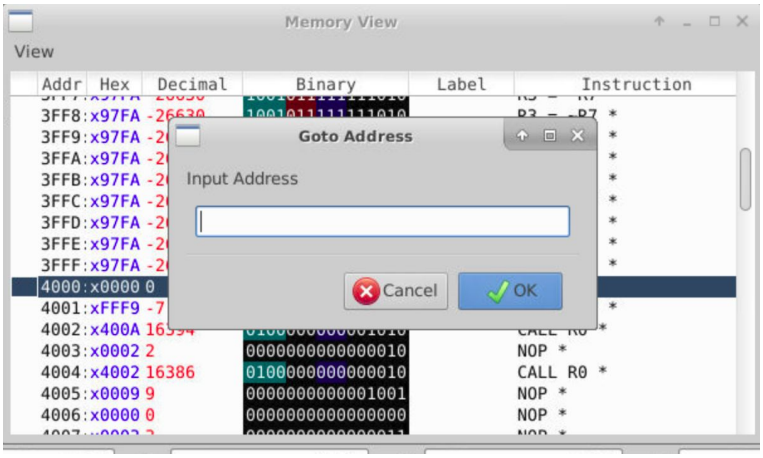


5. Now, complx is set up with the test that you failed! The nicest part of complx is the ability to step through each instruction and see how they change register values. To do so, click the step button. To change the number representation of the registers, double click inside the register box.

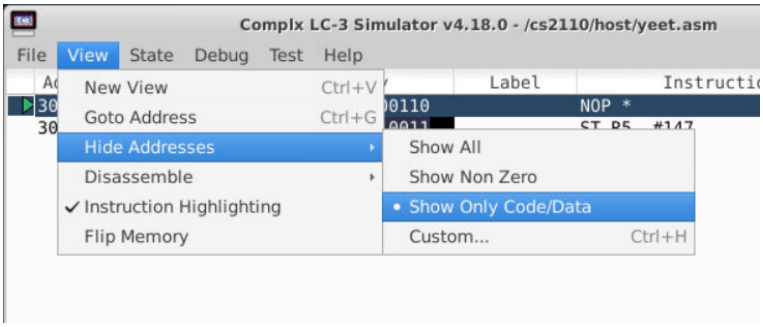
6. If you are interested in looking how your code changes different portions of memory, click the view tab and indicate 'New View'



7. Now in your new view, go to the area of memory where your data is stored by CTRL+G and insert the address



8. One final tip: to automatically shrink your view down to only those parts of memory that you care about (instructions and data), you can use View Tab → Hide Addresses → Show Only Code/Data. Just be careful: if you misclick and select Show Non Zero, it *may* make the window freeze (it's a known Complx bug).



5.3 Appendix C: LC-3 Assembly Programming Requirements and Tips

1. Your code must assemble with **NO WARNINGS OR ERRORS**. To assemble your program, open the file with Complx. It will complain if there are any issues. **If your code does not assemble you WILL get a zero for that file.**

2. **Comment your code!** This is especially important in assembly, because it's much harder to interpret what is happening later, and you'll be glad you left yourself notes on what certain instructions are contributing to the code. Comment things like what registers are being used for and what less intuitive lines of code are actually doing. To comment code in LC-3 assembly just type a semicolon (;), and the rest of that line will be a comment.

3. Avoid stating the obvious in your comments, it doesn't help in understanding what the code is doing.

   Good Comment

   ```
   ADD R3, R3, -1          ; counter--
   BRp LOOP                ; if counter == 0 don't loop again
   ```

   Bad Comment

   ```
   ADD R3, R3, -1          ; Decrement R3
   BRp LOOP                ; Branch to LOOP if positive
   ```

4. **DO NOT assume that ANYTHING in the LC-3 is already zero.** Treat the machine as if your program was loaded into a machine with random values stored in the memory and register file.

5. Following from 3. You can randomize the memory and load your program by doing File - Randomize and Load.

6. Use the LC-3 calling convention. This means that all local variables, frame pointer, etc... must be pushed onto the stack. Our autograder will be checking for correct stack setup.

7. Start the stack at xF000. **The stack pointer always points to the last used stack location.** This means you will allocate space **first**, then store onto the stack pointer.

8. Do NOT execute any data as if it were an instruction (meaning you should put .fills after **HALT** or RET).

9. Do not add any comments beginning with @plugin or change any comments of this kind.

10. **Test your assembly.** Don't just assume it works and turn it in.

5.4 Appendix D: Rules and Regulations

5.4.1 General Rules

1. Starting with the assembly homeworks, any code you write should be meaningfully commented for your benefit. You should comment your code in terms of the algorithm you are implementing; we all know what each line of code does.

2. Although you may ask TAs for clarification, you are ultimately responsible for what you submit. This means that (in the case of demos) you should come prepared to explain to the TA how any piece of code you submitted works, even if you copied it from the book or read about it on the internet.

3. Please read the assignment in its entirety before asking questions.

4. Please start assignments early, and ask for help early. Do not email us the night the assignment is due with questions.

5. If you find any problems with the assignment it would be greatly appreciated if you reported them to the author (which can be found at the top of the assignment). Announcements will be posted if the assignment changes.

5.4.2 Submission Conventions

1. All files you submit for assignments in this course should have your name at the top of the file as a comment for any source code file, and somewhere in the file, near the top, for other files unless otherwise noted.

2. When preparing your submission you may either submit the files individually to Canvas/Gradescope or you may submit an archive (zip or tar.gz only please) of the files. You can create an archive by right clicking on files and selecting the appropriate compress option on your system. Both ways (uploading raw files or an archive) are exactly equivalent, so choose whichever is most convenient for you.

3. Do not submit compiled files, that is .class files for Java code and .o files for C code. Only submit the files we ask for in the assignment.

4. Do not submit links to files. The autograder does not understand it, and we will not manually grade assignments submitted this way as it is easy to change the files after the submission period ends.

5.4.3 Submission Guidelines

1. You are responsible for turning in assignments on time. This includes allowing for unforeseen circumstances. If you have an emergency let us know **IN ADVANCE** of the due time supplying documentation (i.e. note from the dean, doctor's note, etc). Extensions will only be granted to those who contact us in advance of the deadline and no extensions will be made after the due date.

2. You are also responsible for ensuring that what you turned in is what you meant to turn in. After submitting you should be sure to download your submission into a brand new folder and test if it works. No excuses if you submit the wrong files, what you turn in is what we grade. In addition, your assignment must be turned in via Canvas/Gradescope. Under no circumstances whatsoever we will accept any email submission of an assignment. Note: if you were granted an extension you will still turn in the assignment over Canvas/Gradescope.

3. There is a 6-hour grace period added to all assignments. You may submit your assignment without penalty up until 11:55PM, or with 25% penalty up until 5:55AM. So what you should take from this is not to start assignments on the last day and plan to submit right at 11:54AM. You alone are responsible for submitting your homework before the grace period begins or ends; neither Canvas/Gradescope, nor

your flaky internet are to blame if you are unable to submit because you banked on your computer working up until 11:54PM. The penalty for submitting during the grace period (25%) or after (no credit) is non-negotiable.

5.4.4 Syllabus Excerpt on Academic Misconduct

Academic misconduct is taken very seriously in this class. Quizzes, timed labs and the final examination are individual work.

Homework assignments are collaborative, In addition many if not all homework assignments will be evaluated via demo or code review. During this evaluation, you will be expected to be able to explain every aspect of your submission. Homework assignments will also be examined using computer programs to find evidence of unauthorized collaboration.

What is unauthorized collaboration? Each individual programming assignment should be coded by you. You may work with others, but each student should be turning in their own version of the assignment. Submissions that are essentially identical will receive a zero and will be sent to the Dean of Students' Office of Academic Integrity. Submissions that are copies that have been superficially modified to conceal that they are copies are also considered unauthorized collaboration.

You are expressly forbidden to supply a copy of your homework to another student via electronic means. This includes simply e-mailing it to them so they can look at it. If you supply an electronic copy of your homework to another student and they are charged with copying, you will also be charged. This includes storing your code on any site which would allow other parties to obtain your code such as but not limited to public repositories (Github), pastebin, etc. If you would like to use version control, use github.gatech.edu

5.4.5 Is collaboration allowed?

Collaboration is allowed on a high level, meaning that you may discuss design points and concepts relevant to the homework with your peers, share algorithms and pseudo-code, as well as help each other debug code. What you shouldn't be doing, however, is pair programming where you collaborate with each other on a single instance of the code. Furthermore, sending an electronic copy of your homework to another student for them to look at and figure out what is wrong with their code is not an acceptable way to help them, because it is frequently the case that the recipient will simply modify the code and submit it as their own.

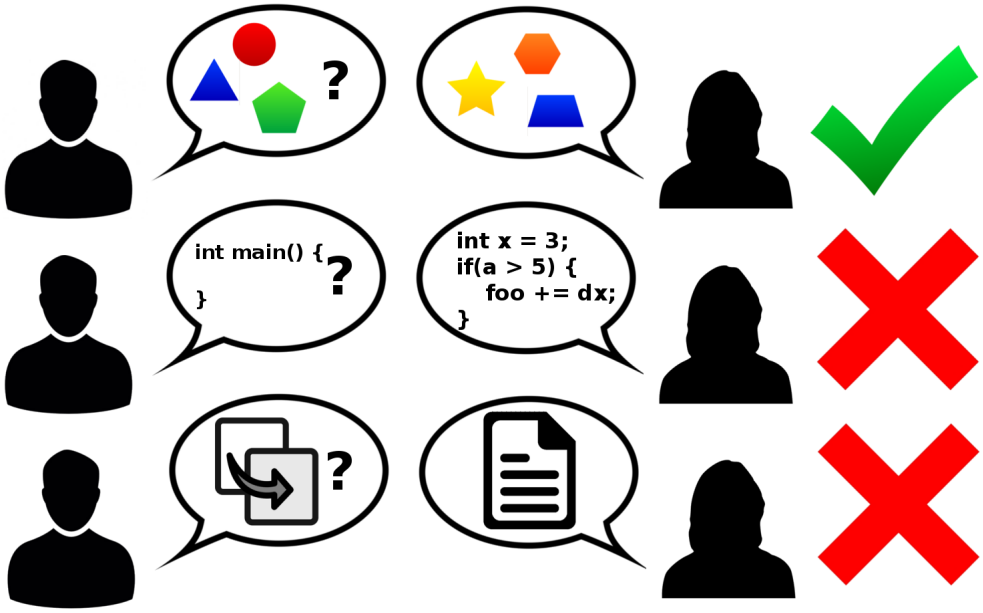


Figure 1: Collaboration rules, explained colorfully