15-122: Principles of Imperative Computation, Fall 2023

Written Homework 1

Due on Gradescope: Monday 4th September, 2023 by 9pm EDT

Name: ______________________________

Andrew ID: ______________________________

Section: ______________________________

This written homework is the first of two homeworks that will introduce you to the way we reason about C0 code in 15-122. It also makes sure that you have a good understanding of key course policies.

Preparing your Submission You can prepare your submission with any PDF editor that you like. Here are a few that prior-semester students recommended:

- **Kami**, **Adobe Acrobat Online**, or ***DocHub***, some web-based PDF editors that work from anywhere.
- ***Acrobat Pro***, installed on all non-CS cluster machines, works on many platforms.
- ***iAnnotate*** works on any iOS and Android mobile device.

There are many more — use whatever works best for you. If you'd rather not edit a PDF, you can always print this homework, write your answers *neatly* by hand, and scan it into a PDF file — *we don't recommend this option, though*.

Caution Recent versions of Preview on Mac are buggy: annotations get occasionally deleted for no reason. **Do not use Preview as a PDF editor.**

Submitting your Work Once you are done, submit this assignment on Gradescope. ***Always check it was correctly uploaded.*** You have unlimited submissions.

If you haven't yet enrolled in Gradescope for this class, please do so by completing the **setup lab** (see Ed or the course web page).

Question:	1	2	3	4	5	Total
Points:	2.5	2	3	3.5	4	15
Score:						

1. **Policies**

BenBhargavCooperLeo

2.5pts

1.1 Collaboration Policy

Read the collaboration policy on the course website. For every statement in each scenario, mark whether it is **OK** or **not OK** according to the collaboration policy. Take each action to be independent from others.

a. Alexander and Alice are on Zoom while completing a written homework.

OK	***not* OK**	
◯	◯	Alexander has already partially completed the homework and keeps it in view on his computer while discussing it with Alice.
◯	◯	Alice screenshares the blank homework writeup so they can both reference it.
◯	◯	Alice asks Alexander to check her work on Task 2 by reading her answer out loud.
◯	◯	Alexander takes notes while brainstorming approaches and uses them to complete the homework later.
◯	◯	Alice draws a diagram on the whiteboard to explain this loop invariant proof.

b. Jackson is trying to work on his programming homework and asks Amanda for advice.

OK	***not* OK**	
◯	◯	Jackson forgets how to transfer files to AFS and Amanda tells him the correct scp command.
◯	◯	Amanda teaches Jackson some vim shortcuts.
◯	◯	Jackson screenshares his code and asks Amanda to help him figure out a bug.
◯	◯	Amanda screenshares the blank homework writeup so they figure out what Task 2 is asking them to do.
◯	◯	Jackson writes out his pseudocode onto a shared whiteboard
◯	◯	Amanda says: "Just use two `for` loops, one for each array. You then create a variable outside of both loops and just compare values to find the overall maximum".
◯	◯	Amanda says: "The writeup said that we want to find the maximum from these two arrays. If you're stuck, I found it helpful to look at the sorting lecture notes."
◯	◯	Jackson asks Amanda to go over his code after both of them have made their final submission, even though grades have not been released. Amanda agrees and walks him through her answers.

c. Mihir and Sophia are studying for a midterm together.

OK	***not*** *OK*	
◯	◯	Sophia screenshares the review slides so that the two of them can figure out a confusing concept from lecture.
◯	◯	Mihir draws out an array on a shared whiteboard to better understand a sorting problem in the practice midterm.
◯	◯	Mihir asks a student from a previous iteration of the course for the midterm from that semester so that he could use the exam as practice.
◯	◯	Mihir asks Sophia how she did a question on a previous, already graded written assignment.
◯	◯	Sophia looks up code for a previous programming assignment on GitHub to figure out how to complete a particularly challenging task.
◯	◯	Mihir takes notes during their study session and uses them when creating his allowed exam notes.

d. Marina is retaking the course.

OK	***not*** *OK*	
◯	◯	To check her work, she looks at the written she did in a previous semester.
◯	◯	Her friend Ethan did very well in the course. When Marina gets stuck on a task, she walks Ethan through her approach.
◯	◯	She just doesn't understand what task 5 of the current programming homework is asking about. She asks her roommate Pranav, who completed the course two years ago, to explain it to her.
◯	◯	VSCode stopped working for her. After closing all her assignment files, she asks her other roommate, Brandon, who also completed the course two years ago, to help her reset it.
◯	◯	During the break before classes starts, Marina goes over all her past assignments that use contracts to make sure she has a good grasp on them this time around, takes detailed notes, and uses them as to complete this semester's assignment.
◯	◯	She is concerned about the upcoming lecture on amortized analysis and reads the lecture notes ahead of time to prepare.

0pts

1.2 MOSS

This part is ungraded but meant to help you think about how MOSS, the software service we use to check for plagiarized code, works. For each statement give your best guess at the right answer. Make sure to come back to it and check the actual correct answers.

a. Robbie is stuck on the last two tasks of the programming homework. Chloe sends him her code for them and he changes all variable names before submitting. Will MOSS detect this?

definitely / very likely / very unlikely / impossible

b. Robbie finds code for the current programming homework on GitHub, modifies it and then submits it. Will MOSS detect this?

definitely / very likely / very unlikely / impossible

c. Robbie asks Pranav, a student who completed the course three semesters ago, for his code for the programming homework. Robbie makes cosmetic changes and submits. Will MOSS detect this?

definitely / very likely / very unlikely / impossible

d. Robbie retakes the course and discovers that the next programming homework is the same as when he took it the first time around. He doesn't look at his old solution, but he is worried that his new code may be too similar to his old code. Will MOSS detect a similarity?

definitely / very likely / very unlikely / impossible

0pts

1.3 Academic Integrity Violation Consequences

This part is ungraded but meant to help you think about the consequences of cheating. For each statement give your best guess at the right answer. Make sure to come back to it and check the actual correct answers.

a. Iliano and Dilsun were caught cheating on task 3 of the current written assignment. Both get reported. What will happen to their grade?

- ◯ they get a zero on task 3
- ◯ they get a zero for the whole assignment
- ◯ they get a negative grade for the whole assignment
- ◯ they fail the course

b. Brandon, a student from a previous semester, got caught giving his code to Iliano who is currently taking the course. What will happen to Brandon?

- ◯ nothing
- ◯ he gets reported and nothing else
- ◯ he gets reported and is given a symbolic zero on that assignment
- ◯ he gets reported and his course grade is lowered by one letter grade
- ◯ he gets suspended

c. Iliano was reported for an academic integrity violation in a history course last semester and is being reported again this semester for copying code. What will happen to him?

- ◯ nothing besides the new report
- ◯ he will appear in front of an academic review board (ARB) and be given a stern lecture
- ◯ he will appear in front of an ARB and most likely be suspended
- ◯ he will appear in front of an ARB and most likely be expelled

0pts

1.4 Academic Integrity Contract

Now that you had a chance to reflect on the collaboration policy of the course, we ask you to complete and sign the contract on the next page. By doing this, you declare that you understand the course policy on academic integrity and commit to abide by it. **Like any contract, read it carefully.** Please reach out to the course staff if you have any questions.

Although this task is worth 0 points, failure to complete and sign the contract will carry a penalty of **-500 points**, i.e., guaranteed failure in the course.

15–122 — Principles of Imperative Computation, Fall 2023

The value of your degree depends on the academic integrity of yourself and your peers in each of your classes. It is expected that, unless otherwise instructed, the work you submit as your own will be your own work and not someone else's work or a collaboration between yourself and other(s).

Please read carefully the academic integrity policy of this course and the University Policy on Academic Integrity carefully to understand the penalties associated with academic dishonesty at Carnegie Mellon. In this class, cheating/copying/plagiarism means copying all or part of a program or homework solution from another student or unauthorized source such as the Internet, giving such information to another student, having someone else do a homework or take an exam for you, reusing answers or solutions from previous editions of the course, or giving or receiving unauthorized information during an examination. In general, **each solution you submit (quiz, written assignment, programming assignment, midterm or final exam) must be your own work**. In the event that you use information written by another person in your solution, you must cite the source of this information (and receive prior permission if unsure whether this is permitted). It is considered cheating to compare complete or partial answers, copy or adapt others' solutions, read other students' code or show your code to other students, or sit near another person who is taking the same course and complete an assignment together. Writing code for others to see (e.g., on a whiteboard) is never permitted. It is also considered cheating for repeating students to reuse their solutions from a previous semester, or any instructor-provided sample solution.

It is a violation of this policy to hand in work for other students.

Your course instructors reserve the right to determine an appropriate penalty based on the violation of academic dishonesty that occurs. *Penalties are severe: a typical violation of the university policy results in the student failing this course, but may go all the way to expulsion from Carnegie Mellon University.* If you have any questions about this policy and any work you are doing in the course, please feel free to contact your instructors for help.

We will be using the MOSS system to detect software plagiarism.

By checking the second box below, you commit to performing a chicken dance in front of the TAs at office hours. Most people do not check this box.

It is not considered cheating to clarify vague points in the assignments, lectures, lecture notes, or to give help or receive help in using the computer systems, compilers, debuggers, profilers, or other facilities, but you must refrain from looking at other students' code while you are getting or receiving help for these tools. It is not cheating to review graded assignments or exams with students in the same class as you, but it is considered unauthorized assistance to share these materials between different iterations of the course. **Do not post code from this course publicly (e.g., to Bitbucket or GitHub).**

☐ I have read the statements above and reviewed the course policy for cheating and plagiarism.

☐ I agree to the clause in paragraph 6.

By signing below, I commit to abiding by these policies in this course.

Andrew ID ____

Name (print) ____ Section __________________

Signature ____ Date __________________

2. **Running C0 Programs**

Assume we have the files `bar.c0` and `bar-test.c0`. The file `bar.c0` contains a function `bar` that takes an integer argument and returns an integer. The file `bar-test.c0` contains this `main` function (and nothing else):

```
int main() {
  int x = bar(492023);
  return x;
}
```

How to run this program? Check out a relevant page in the C0 Tutorial at `https://bitbucket.org/c0-lang/docs/wiki/Tutorial` and answer the following questions.

1pt **2.1** From the command line, show how to display the value returned by `bar(492023)` using the C0 compiler.

1pt **2.2** From the command line, show how to display the value returned by `bar(492023)` using the C0 interpreter.

3pts

3. Preconditions and Postconditions

For the following functions, **either** check the box that says the postcondition always holds when the function is given inputs that satisfy its preconditions **or** give a concrete counterexample: specific values of the inputs such that the preconditions (if there are any) holds and the postcondition does not hold. You don't have to write any proofs.

```
int f1(int x, int y)
//@requires 0 < x && x <= y;
//@ensures \result < 0;
{
  return x - y;
}
```

@ensures always true? ☐

x = ☐ y = ☐

```
int f2(int x)
//@requires x % 2 == 0;
//@ensures x >= 0 || \result > x;
{
  return x / 2;
}
```

@ensures always true? ☐

x = ☐

```
int f3(int x, int y)
//@requires y < -1;
//@ensures \result > y ;
{
  return x % y;
}
```

@ensures always true? ☐

x = ☐ y = ☐

```
int f4(int x, int y)
//@requires x + y == 42;
//@ensures \result - x == y;
{
  return 42;
}
```

@ensures always true? ☐

x = ☐ y = ☐

```
int f5(int x, int y)
//@ensures \result > 0;
{
  if (x < 0) x = -x;
  if (y < 0) y = -y;
  if (y > x) {
    return y - x;
  } else {
    return x - y;
  }
}
```

@ensures always true? ☐

x = ☐ y = ☐

```
int f6(int x, int y)
//@ensures \result <= 0;
{
  if (x <= 0) x = -x;
  if (y <= 0) y = -y;
  if (y/2 >= x/2) {
    return x - y;
  } else {
    return y - x;
  }
}
```

@ensures always true? ☐

x = ☐ y = ☐

4. **Thinking about Loops**

When we think about loops in 15-122, we will always concentrate on a single arbitrary iteration of the loop. A loop will almost always modify something; the following loop modifies the local variable `i`.

```
while (i < n) {
   i = i + 4;
}
```

In order to reason about the loop, we have to think about the two different values stored in the variable `i` during an iteration. We use the variable i to talk about the value stored in the variable `i` before the loop runs (before the loop guard is checked for the first time). We use the "primed" variable i' to talk about the value stored in the variable `i` after the loop runs exactly one more time (before the loop guard is next checked).

An important part of figuring out what a loop does is to understand how it variables are updated, i.e., what the value of the prime variables are. This question is about this. Write all your answers in simplified form.

1pt **4.1** Consider the following loop:

```
while (a < n) {
  c = 2 * b - c;
  b = 3 * b + a;
  a = a - 1;
}
```

- If $a = 6$, $b = 2$, and $c = 7$, then assuming $6 < \texttt{n}$,

  a' = ______, b' = ______, and c' = ______

- If $a = 3y$, $b = x + y$, and $c = 2y$, then assuming $3y < \texttt{n}$, in terms of x and y,

  a' = ______, b' = ______, and c' = ______

- If $b = c$, then assuming $a < \texttt{n}$, in terms of a and c,

  a' = ______, b' = ______, and c' = ______

- In general, assuming $a < n$, then in terms of a, b, and c,

  a' = ______, b' = ______, and c' = ______

Note that we always say "assuming (something) $< \texttt{n}$," because if that were not the case the loop wouldn't run, and it wouldn't make any sense to be talking about the values of the primed variables.

1pt

4.2 Consider this loop:

```
while (...) {
  a = a + 3;
  b = b * 2 + a;
  c = c + a - b;
}
```

Be careful, it looks similar but is trickier! Give simplified answers.

- If $a = 7$, $b = 3$, and $c = 9$, then assuming the loop guard evaluates to true,

  $a' =$ [ ], $b' =$ [ ], and $c' =$ [ ]

- In general, assuming the loop guard evaluates to true, then in terms of a, b, and c,

  $a' =$ [ ] $b' =$ [ ], and $c' =$ [ ],

1.5pts

4.3 Consider this loop:

```
while (a > 0 || b > 0) {
    if (a < b) {
        b = b-a;
    } else {
        a = a-b;
    }
}
```

- If $a = 42$ and $b = 17$, then

  $a' =$ [ ] and $b' =$ [ ]

- If $a = x + y$ and $b = x$, where x is a small positive integer and y is a small non-negative integer, then

  $a' =$ [ ] and $b' =$ [ ]

- If $a = x$ and $b = x + z$, where x and z are both small positive integers, then

  $a' =$ [ ] and $b' =$ [ ]

- If $a > 0$ and $b > 0$, one of the two cases above will always be true. Therefore, we can conclude which of the following about the values stored in `a` and `b` after an arbitrary iteration of the loop? (Check **all** that apply)

  - [ ] $a' \geq 0$ and $b' \geq 0$
  - [ ] $a' > 0$ and $b' \geq 0$
  - [ ] $a' \geq 0$ and $b' > 0$
  - [ ] $a' > 0$ and $b' > 0$

5. **Proving a Function Correct**

In this question, we'll do part of the proof of correctness for a function `compute_sum` relative to a specification function `SUM`. You may assume that the loop invariants have already been proved to be valid.

```
int compute_sum(int n) {
  int total = 0;
  while (n > 0) {
      total = total + n;
      n = n - 1;
  }
  return total;
}
```

1pt

5.1 Complete the specification function below with the simple mathematical formula that gives the sum of numbers from 0 to n.

```
int SUM(int n)
//@requires 0 <= n && n < 94230;
{
  return ______________________________________________;
}
```

Give a postcondition for `compute_sum` **using this specification function**.

```
int compute_sum(int num_ints)
//@requires 0 <= num_ints && num_ints < 94230;

//@ensures ____________________________________________;
{
  int n = num_ints;
  int total = 0;
  while (n > 0)
  //@loop_invariant 0 <= n;
  //@loop_invariant n <= 94230;
  // Additional loop invariant will go here
  {
      total = total + n;
      n = n - 1;
  }
  return total;
}
```

Note: in the real world we wouldn't have an efficient closed-form solution used as a specification function for an inefficient loop-based solution. We usually use the slow, simple version as the specification function for the fast one!

0.5pts

5.2 Had we not introduced the local variable `n` on line 12 and used `num_ints` instead, `compute_sum` would not compile. Explain why.

Why was it necessary to add the new local `n` in the second version of `compute_sum`? *Hint: try compiling this code.*

1.5pts

5.3 Using `SUM` everywhere possible, give a suitable extra invariant that would allow us to prove the function correct. *(Consider creating a table with values that change during the loop.)*

```
17 //@loop_invariant ______________________________;
```

Which line numbers would we point to to justify that `n == 0` after the loop?

Substitute in 0 for `n` in your loop invariant on line 17 and then simplify.

If your last answer is correct, substituting `\result` for `total` in the simplified version will yield exactly the postcondition on line 10. This proves that the loop invariants and the negation of the loop guard imply the postcondition.

1pt

5.4 Termination arguments for loops (in this class) must have the following form:

During an arbitrary iteration of the loop, the expression ___...___ gets strictly ___larger / smaller___, but this expression can't get ___larger / smaller___ than ___...___ on which the loop guard is false.

Why does the loop in `compute_sum` terminate?

During an arbitrary iteration of the loop, the expression ______________________ gets strictly ___larger / smaller___, but this expression can't get ___larger / smaller___ than __________ on which the loop guard is false.