Lab 6

Due Monday, April 8 by 11:59pm

Handout 17
CSCI 334: Spring 2024

Coding Guidelines

Each question in this assignment must be written using LaTeX. I provide a LaTeX template in your repository for you to use to get started.

Keep in mind that LaTeX is a programming language. The template I provide compiles without error as-is. Treat this homework as you would with any other programming language: make small changes and compile frequently using `pdflatex`. For full credit, you must submit both your `.tex` source file as well as the rendered `.pdf` file. Your source file should be called `lab-6.tex` and your PDF should be called `lab-6.pdf`. (5 points) Your source file should compile without error. (5 points)

Turn-In Instructions

Turn in your work using the `git` repository assigned to you. The name of the `git` repository will have the form `https://aslan.barowy.net/cs334-s24/cs334-lab06-<USERNAME>.git`. For example, if your CS username is `abc1`, the repository would be `https://aslan.barowy.net/cs334-s24/cs334-lab06-abc1.git`.

You should have received an invite to commit to the repository via email. If you did not receive an email, please contact me right away!

Group Programming Assignment

This is a partner lab. You may work with another classmate if you wish, and you may co-develop solutions. Remember: although you can work on code together, you must each independently write up and submit your solution. No code copying is allowed. Tell me who your partner is by committing a `collaborators.txt` file to your repository. **Be sure to commit this file whether you worked with a partner or not.** If you worked by yourself, `collaborators.txt` should contain something like "I worked by myself." (5 points)

This assignment is due on Monday, April 8 by 11:59pm.

Reading

1. **(Required)** "How to Fix a Motorcycle"
2. **(Required)** "Proof by Reduction"

Problems

Q1. (15 points) ..... How to Fix a Motorcycle

Read the excerpt of Zen and the Art of Motorcycle Maintenance titled "How to Fix a Motorcycle" by Robert Pirsig, and then answer the following questions. You may write as much as you like, but in the interest of keeping it feasible for me to read all of these, I would appreciate it if you kept your responses short. A good guideline is to write no more than a sum total of 400 words for all three questions.

(a) Think back about your experiences doing programming in the past. Think of a time when it was difficult and you were feeling demoralized. The experience may have been a programming assignment, programming for work, or programming for fun. Describe your experience and how any of Pirsig's "gumption traps" may apply.

(b) Whether Pirsig's advice pertains to your situation or not, what would you do differently now if you were in that situation again?

(c) To what extent do you think your situation could have been improved by a better programming language? In other words, suppose your programming language helped you more. What pitfalls or gumption traps might be avoided? Don't worry too much about whether your imagined features are impractical to implement or even impossible (e.g., not computable).

Although this assignment is a partner lab, the answers to this particular question should be your own.

Q2. (30 points) ..... Partial and Total Functions

For each of the following function definitions, (i) give the graph of the function. Say whether this is a (ii) partial function or a total function on the integers. If the function is partial, say where the function is (iii) defined and where it is (iv) undefined. You may limit your analysis to determining the set of integers for which the function is defined.

For example, take the function $\texttt{f(x)} = \texttt{if x} > 0 \texttt{ then x} + 2 \texttt{ else x/0}$

The graph of this function is the set of ordered pairs $\{\langle x, x+2\rangle \mid x > 0 \wedge x \in \mathcal{Z}\}$. The function is partial. It is defined on all integers greater than 0 and undefined on integers less than or equal to 0.

Functions:

(a) $\texttt{f(x)} = \texttt{if x} < 10 \texttt{ then 0 else f(x} - 2\texttt{)}$

(b) $\texttt{f(x)} = \texttt{if x} + 3 > 3 \texttt{ then x} + 4 \texttt{ else x/0}$

(c) $\texttt{f(x)} = \texttt{if sin(x)} > 0 \texttt{ then 1 else f(x} + \pi\texttt{)}$

Q3. (20 points) ..... Halting on any input

The function $\texttt{Halt}_{\texttt{Any}}$ is defined as:

$$\texttt{Halt}_{\texttt{Any}}(\texttt{p}) = \begin{cases} \texttt{true} & \text{if p halts on any input.} \\ \texttt{false} & \text{otherwise.} \end{cases}$$

Assume `p` is a `String` representation of a Python function always having the form:

```
def prog(x):
    # whatever
```

and that `x` is just an ordinary primitive value, like an integer.

Prove that $\texttt{Halt}_{\texttt{Any}}$ is not computable. Your answer should be in the form of a function, written in the programming language of your choice.

Note: do not confuse $\texttt{Halt}_{\texttt{Any}}$ with $\texttt{Halt}_{\texttt{All}}$. The function $\texttt{Halt}_{\texttt{All}}$ halts when all inputs cause some program, `p`, to halt. For example, for the following program, $\texttt{Halt}_{\texttt{Any}}$ returns `true` while $\texttt{Halt}_{\texttt{All}}$ returns `false`:

```
def prog(i):
  if i = 0
    return
  else
    while true
```

Q4. (25 points) ..... Garbage Collection

A garbage collection algorithm performs automatic cleanup of unused memory in a program. Modern programming language runtimes routinely perform garbage collection in order to dramatically simplify memory management. Garbage has the following definition.

> At a given point i in the execution of a program P, a memory location m is garbage if continued execution of P from i **will not** access location m again.

Nonetheless, garbage collection using the above definition of garbage is not computable. Instead, languages solve a simpler problem by using a slightly different definition of garbage:

> At a given point i in the execution of a program P, a memory location m is definitely garbage if continued execution of P from i **cannot** access location m again.

John McCarthy, inventor of the LISP programming language, also invented garbage collection. His "mark sweep" algorithm uses the latter definition, because it only reclaims memory that is impossible to re-read.

Prove that garbage collection using the first definition is not computable by reduction. Your answer should be in the form of a function, written in the programming language of your choice.

Assume that you have the following `isGarbage` function available in the standard library of the programming language of your choice.

```
boolean isGarbage(String p, String m, int i)
```

Calling `isGarbage` with the source code for program text `p`, variable name `m`, and line number `i` has the following behavior.

`isGarbage(p, m, i)`	returns `true` if `m` is garbage at line `i` of program `p`.
`isGarbage(p, m, i)`	returns `false` otherwise.

You may assume that `isGarbage` always halts. You may also assume that `p` is "simple" code that does not contain class or function definitions.

Hint: we know that `Halt`, $\texttt{Halt}_{\texttt{All}}$, and $\texttt{Halt}_{\texttt{Any}}$ are not computable. Take your pick.

Q5. ($\frac{1}{10}^{\text{th}}$ bonus point) ..... Optional: Feedback

I always appreciate hearing back about how easy or difficult an assignment is.

For $\frac{1}{10}^{\text{th}}$ of a bonus to your final grade, please fill out the following Google Form.