Homework 1: Warm-up
Due: 10:59 PM, Sep 11, 2019

Contents

1 CS 17 Course Policies (27 points) 2
2 Prefix (10 points) 3
3 Racket Syntax (8 points) 4
4 BNF (8 points) 5
5 Arithmetic Expressions (8 points) 5
6 Palindrome (Optional) 7

Objectives

By the end of this homework, you will understand:

1. Course policies
2. Some Racket syntax
3. BNF (Backus-Naur Form) notation

This homework is a little unusual in that it involves some reading (the course policies).

How to Hand In

You will hand in this homework by placing your answers within various txt files.

A txt file is a plain text file (that is, one that doesn’t contain any images or formatting). Most text editors (Microsoft Word, for example) will not save your documents in the .txt format by default, so be sure to select the “Save as” option and select .txt when saving. On a Mac, the TextEdit program is great for creating text-only files, but when you’ve opened a new document, immediately select Format ... Make Plain Text. (If only Make Rich Text is available, then you needn’t do anything: your preferences have been set to create plain text files, which is good.)

Specifically in this assignment, you will submit the following files:

- policies.txt
Problems

1 CS 17 Course Policies (27 points)

Task: The CS 17 course missive, collaboration policy, as well as guides to TA hours and pair programming can all be found on the course website. For each of the following situations involving fictional CS 17 students, decide whether their actions are appropriate per the course policies, and explain why or why not.

Note: The response for the first situation is provided to give you an example of the level of explanation expected in your responses. For some of these questions, the correct answer is to state that the situation is allowable under course policy.

1. To get closer to her classmate Petunia the pig, Quinn the duck decides to work on a lab with her. As they get settled, Petunia mentions that she’d prefer to drive. “What a coincidence!” Quinn exclaims, as he says that he prefers navigating because his webbed feet slip and slide over the keys, making him slow at typing. The two know it’s important to get experience in both roles, so they decide to spend 30 minutes in their preferred role and 15 in the other as they pair program the lab.

   This violates the collaboration policy. The expectation is that partners pair program as outlined in the pair programming handout. This includes spending an equal amount of time in both roles.

2. Bessie the cow and her friend Clarence the alpaca partner up for the Rackette project. They enjoy working on the design check together, and decide to pair program the code. Bessie then sees an opening in her farm’s fence and bolts for it, gets distracted by a yummy grass field, takes a nap, and forgets her way back. She misses a few meetings. She then misses another scheduled meeting because an bucket of lettuce was left too close to her pasture. Clarence doesn’t want to fall behind so he codes a few parts of the project himself. Bessie makes it to their last two meetings, and they finish up and submit their project together.

3. Fernando the domestic yak and Raj the goat partner up for Lab 06. Fernando missed a recent lecture because of a check-up with his owner, so he asks Raj if he minds if they look over the notes together before working on the assignment. After five minutes or so, they get to work.
When Fernando driving their teamwork, he takes Raj’s suggestions and flips to the lecture notes as necessary to make sure he understands everything. When Fernando is navigating, he asks Raj to explain code that’s unclear.

4. Harriet the horse was having a bad week because it rained every day, and she couldn’t go outside. She had put off working on HW 8 until the very last minute . . . but then she remembered she had a horse show to attend. The next day, she finished the homework and turned it in late, hoping for partial credit and figuring that one day late is better than never.

5. Lien the llama is having some serious problems with this week’s homework. While sitting in the Sun Lab aimlessly browsing through directories, she happens to discover that Joey the sheep-dog has world-readable permissions set on his course folder. Excited that she can now read the contents, Lien finds that Joey hasn’t yet completed the homework, but has completed the first problem, and written some notes about possible approaches to the remaining problems in code comments. Lien exits out of the folder and does the homework on her own.

6. Sophie the milkmaid is having trouble with a homework problem, so she asks Elijah the farmer for help. Elijah offers to meet her in the SciLi study center. They meet up and, with Elijah leading most of the discussion, come up with a solution on the white board. Afterwards, they erase the white board and head to their own rooms to write up their solution.

7. Chico the chicken bumps into Michael the rooster at the V-Dub during the final exam. Chico mentions that the last problem is a real pain. Michael tells him that he disagrees, but that they’re not supposed to talk about the exam and he’d rather not discuss it further.

8. Buster the kunekune has had a difficult time trying to get help at hours this week, because everyone there keeps laughing at his squishy face and snout. He sees a TA at the Ratty during lunch and asks her a question about the homework, figuring that he should take the opportunity to ask for help without having to wait in line and endure the ridicule.

9. Mahdi the mouse, hiding in the walls of Farmer Spike’s barn, loses track of what day it is. He comes out of the wall and realizes that his first project is due extremely soon, but it’s way too late to work on it with his partner. He thinks about emailing Spike for an extension, but decides to email the TA Staff to ask instead because he feels bad about hiding in Spike’s walls.

10. During Lab 02, Gabriela the goose and Sarah the ox work together and become good friends, bonding over their shared ex-boyfriend, Christopher the donkey. They meet up the next day in the Ratty and reminisce about how great of a partnership they had. Gabriela understands Sarah in ways that Kaylah the sheep, her partner during Lab 01, never could! They can’t wait until the next lab session, when they will have a chance to work together again! When their next lab finally rolls around, they make a beeline for the first open computer and start working together.

Task: Fill out the collaboration policy Google Form if you have not already done so in Lab 1! We will not grade any of your further assignments until you have filled out this form.
2 Prefix (10 points)

Task: Convert the following algebraic expressions using infix notation (which you know from algebra class, where the operator is between the operands, like $3 + 5$) into Racket expressions using prefix notation (notation where the operator precedes the operands, like $(+ \ 3 \ 5)$). We’ve parenthesized the expressions for you so that there’s no ambiguity about the order of operations.

Hint: No two of these expressions have quite the same translation!

1. $(4 \times 3) + 2$
2. $4 \times (3 + 2)$
3. $(4 - 1) + (3 \div 2)$
4. $4 - ((1 + 3) \div 2)$
5. $(4 - (1 + 3)) \div 2$
6. $((60 \div 5) - 6) + (2 \times 4)$
7. $(60 \div (5 - 6)) + (2 \times 4)$
8. $(60 \div 5) - ((6 + 2) \times 4)$
9. $((60 \div 5) - (6 + 2)) \times 4$
10. $60 \div (5 - ((6 + 2) \times 4))$

3 Racket Syntax (8 points)

We’ve given you some rules for CS17 Racket (the tiny subset of Racket we’ll be using for the next month). If you choose the "#lang Racket" language in Dr. Racket, then many of the following programs will run fine.

Ex.

```
(define sandy-is-a-cat true)
(define barnyard-height 13.6)
(define cow "moo")
(define bessie "cow")
```

However, several of the programs are not allowable CS17 Racket.

Task: For each program that’s not allowed, give at least one reason why it’s not. You may find it helpful to refer to the Racket Style Guide on the course website. Please do not use DrRacket to evaluate these expressions!

1.

```
(define a 3)
(define a 4)
(+ a a)
```
2.

```scheme
(define a_is_my_number true)
```

3.

```scheme
(define 13by4 52)
```

4.

```scheme
(define pig true)
(define .pig (+ pig pig))
```

4 BNF (8 points)

We used a little bit of BNF to describe what was an OK CS17 Racket program. So far it looks like this:

```plaintext
<program> := <definition>* <top-level-expression>
<top-level-expression> := <expression>
<expression> := <boolean> | <number>
<boolean> := true | false
<number> := anything that looks like a number
```

Which accounted for a lot of the stuff we played with in class. The code “(+ 17 18)” doesn’t fit this description, but we’ll soon expand the description to allow that as yet another kind of expression. We even played with a tiny BNF for a much simpler language consisting of strings of letters like A, B, C. Here’s another one of those:

```plaintext
<text> := <capital><lower>*
<capital> := A | B | C | D
(lower> := a | e | n | r | s | t
```

(yes, there really are only four capital letters allowed!)

For this tiny language, the “tokens” are certain individual letters, and blanks aren’t even part of the language, so every legal “text” consists of just a sequence of letters.

Task: For each of the following, tell whether it’s an allowable “text” according to the BNF description above:

1. stern
2. B
5 Arithmetic Expressions (8 points)

Here’s a BNF for a limited class of arithmetic expressions in CS17 Racket:

To keep things simple, we’ve made “number” into something simple-to-write-down, so that we can have a complete BNF. A “number” here is what you’d call a one-digit-number in ordinary English.

```
<arithmetic-expression> ::= <number> | (<operation>_<operand>_<operand>)
<operation> ::= + | - | * | /
<operand> ::= <arithmetic-expression>
<number> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

**Formatting Note:** The tokens in this language are parentheses, individual blanks (indicated by an underscore in the BNF, to make them visible!), the four operations, and the ten individual digits.

**Task:** Which of the following are legal arithmetic operations, according to the BNF above? For any that are not legal, explain why in a brief sentence.

**Example:**

22 isn't legal because 22 isn't one digit

1. 

3 + 2

2. 

(3 + 2)

3. 

(+ 4 5)

4. 

6
5. 
\((- (9 * 2) (3 / 1))\)

6. 
\((/ 7 (* 2 2))\)

7. 
\(((* 4 3) (\-) (/ 6 2))\)

8. 
\((* (/ 5 2) (\- 9 1))\)

6  Palindrome (Optional)

A palindrome is a sequence of characters that’s the same backwards as forwards. Sometimes blanks, capitalization, and punctuation are ignored, making things like ”Doc, note: I dissent. A fast never prevents a fatness. I diet on cod” get counted as valid palindromes. In computer science, palindromes don’t have to form words, so ASHDKJKDHS is a perfectly good palindrome.

Let’s keep things simple and stick with the letters ABCD only. Legal palindromes with these characters would be things like B, ACA, ACCA, ADADDADA, etc. Non-palindromes would be ACAD, ABD, DADA, etc.

**Task:** Try to write out BNF that starts with:

```
<palindrome> := ...
```

and ends with:

```
<letter> := A | B | C | D
```

with the property that every finite string of the letters ABCD that is a palindrome actually matches your BNF description, and every non-palindrome using those four letters does not match.

**Note:** In this question, we’re ignoring spaces - there should be no spaces in any strings at all.

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS 17 document by filling out the anonymous feedback form: [http://cs.brown.edu/courses/csci0170/feedback](http://cs.brown.edu/courses/csci0170/feedback).