Check-In One Summary
CS 364 — Spring 2021

Overview

Student Check-Ins are brief individual meetings for you to ask clarifying questions about the material we've covered in the course and for me to check that you are understanding key ideas. These meetings will offer you a structured way to reflect on the material you've retained as well as content that might still be a bit confusing.

In addition to the individual meetings, you will also turn in a written summary of the topics covered in class. I will provide you with a set of guiding questions for these summaries. The purpose of the written summary is to help you review material and identify aspects of the class that are challenging and/or confusing.

Individual Meetings (25 points)

Individual student check-in meetings will take place **Thursday February 18**. There will be no class meetings taking place during this week. Student hours will still be available at their standard times.

Use the sign-up tool on Sakai to sign up for **one twenty minute time slot**. All individual meetings will be hosted using the **Student Hours meeting found on Sakai**. Please be prompt when joining the Zoom meeting at your specified time. There is a waiting room, so you may join early.

Please contact me ASAP if you are unable to schedule a meeting in the available times.

Failure to sign up for a meeting and/or show up on time for your meeting slot may result in a reduced grade.

Written Summary (75 points)

Due: Wednesday February 17 at 11:59 PM Eastern

Submit your written summary to Gradescope by the deadline specified above. To give me time to review your answers prior to the check-in meetings, this is a strict deadline. Please plan your time accordingly. You may either type up your answers or hand-write your answers and scan/take pictures to submit your summary. Space is provided in this outline to record your answers, should you wish to print out a copy of this document.

Collaboration and Resource Policy

You **may** use your notes, class videos, lab assignments, and assigned readings to help you with this summary. **You may not use any other resources**. This summary should be completed **individually**. Please review the academic integrity and professionalism policy in the course syllabus for additional details.

1 Functional Programming (20 points)

1.1 Definitions and Background

1. Define the following terms and give examples where appropriate.
   (a) binding:

   (b) lambda expression:

   (c) variant type:

(d) first class function:

(e) higher-order function:

(f) closure:

(g) referential transparency:

2. What are some differences between programming languages? Provide several concrete examples.

3. Briefly describe Imperative, Object-Oriented, Functional, and Declarative programming paradigms. What some typical characteristics of each?

1.2 Recursion and Function Evaluation

1. Write a recursive function called `power` that inputs two non-negative integers `x` and `y` and outputs $\mathtt{x}^{\mathtt{y}}$ using multiplication.

```
let power = (x:int, y:int) : int =>
```

2. Evaluate the following expressions, showing several steps on the way to the final value.

(a)
```
let example = (x, y) => { abs(x - y) };
example( 4, 8 );
```

(b)
```
List.filter( x => { x mod 2 == 0 },
                  List.map( x => { x + 3 }, [ 1, 2, 4, 5, 6, 10 ] ));
```

1.3 Higher-Order Functions

Consider the following function definition for `fold2`, which folds over two, equal-length lists:

```
let rec fold2 = ((f: 'a => 'b => 'c => 'a), (acc:'a), (l1:list('b)), (l2:list ('c))) : 'a => {
    switch( (l1, l2) ) {
        | ([],[]) => acc
        | ([hd1, ...tl1], [hd2, ...tl2]) => fold2(f, f(acc, hd1, hd2), tl1, tl2)
        | _ => failwith("lists␣have␣different␣lengths")

    };
};
```

This function can be used to implement other higher-order functions. Demonstrate this ability by implementing **ONE OF** the following function using `fold2`.

1.
```
/*
 * Given f, [a1, ..., an], [b1, ..., bn]
 * return [ f(a1, b1), ..., f(an, bn) ]
 */
let map2 = ( (f: 'a => 'b => 'c), (l1: list('a)), (l2: list('b)) : list('c) =>
```

2.
```
/*
 * Given f, [a1, ..., an], [b1, ..., bn]
 * return true if f(ai, bi) returns true for all 1 <= i <= n
 */
let for_all2 = ((f: 'a => 'b => bool), (l1: list('a)), (l2: list('b)) : bool =>
```

2 Interpreters and Compilers (10 points)

1. What are the stages of an interpreter? What data types are passed between these stages? What tools are used to generate some of these stages?

2. What differences are there between a compiler and an interpreter?

3 Lexing, Regular Expressions, and Automata (20 points)

3.1 Definitions

Define the following terms and give examples where appropriate.

1. lexeme:

2. token:

3. alphabet:

4. language over an alphabet:

5. regular language:

6. maximal munch rule:

7. lexical analyzer generator:

8. deterministic finite automaton:

9. nondeterministic finite automaton:

10. finite automaton acceptance:

3.2 Regular Expressions and Automata

1. Write a regular expression to match each of the following.
   - An RGB color: three comma-separated integers enclosed in parentheses
   - A Java variable name: a sequence of lowercase letters, upper case letters, numbers and underscores that does not begin with a number.

2. How can a character class be represented using only single match (a), empty match (ε), concatenation (AB), union (A|B), and Kleene star (A*)?

For this question, regular expressions may be single character (a), epsilon (ε), concatenation (AB), union (A|B), Kleene star (A*), plus (A+), and option (A?).

1. Write a regular expression (over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$) for the language of strings that: (a) begins with *EITHER* a followed by b followed by zero or more repetitions of this sequence *OR* c followed by d followed by zero or more repetitions of this sequence, (b) concludes with a nonzero, even number of d, and (c) does not contain any other letters between parts (a) and (b). Use at most 20 symbols in your answer.

2. (7 points) Draw an NFA (or DFA) that accepts the language from the above problem. Use at most fifteen states in your answer.

4 Context-Free Grammars and Parsing (20 points)

4.1 Definitions

Define the following terms and give examples where appropriate.

1. Context-Free Grammar:

2. Non-terminal:

3. Terminal:

4. Language of a Context-Free Grammar:

5. Derivation:

6. Parse Tree:

7. Ambiguous Grammar:

8. Left-Recursive Grammar:

4.2 Context-Free Grammars

1. Let L_1 be the language consisting of all non-empty palindromes over the alphabet $\Sigma = \{\texttt{a}, \texttt{b}\}$. That is, L_1 consists of all sequences of a's and b's that read the same forward or backward. For example, `aba` $\in L_1$ and and `bb` $\in L_1$ and `aabbbaa` $\in L_1$, but `abb` $\notin L_1$. Write a context-free grammar for L_1.

4.3 Ambiguity

1. (6 points) Consider the following grammar G_1:

$$\begin{array}{lcl}
S & \rightarrow & E \\
E & \rightarrow & \texttt{if}\ E\ \texttt{then}\ E\ \texttt{else}\ E \\
E & \rightarrow & \texttt{if}\ E\ \texttt{then}\ E \\
E & \rightarrow & \texttt{int} \mid \texttt{id} \\
E & \rightarrow & E\ \texttt{<}\ E
\end{array}$$

Assume that any numbers become `int` tokens and any unbound strings become `id` tokens. Show that this grammar is ambiguous using the string:

"`if x < 3 then if 12 < y < 24 then y else x`"

2. (10 points) Consider the following grammar G_2:

$$\begin{array}{lcl} E & \rightarrow & E + T \mid T \mid E\texttt{!} \\ T & \rightarrow & \texttt{int} \mid \texttt{(}E\texttt{)} \end{array}$$

Rewrite the following grammar to eliminate left recursion. That is, provide a grammar G_3 such that $L(G_2) = L(G_3)$ but G_3 admits no derivation $X \longrightarrow^* X\alpha$.

4.4 Earley Parsing

1. Complete the Earley parsing table using the following grammar and input token stream. Does the token stream parse? How do you know?

Grammar

$$\begin{array}{rcl} S & \rightarrow & \texttt{id}\ A\ M \\ A & \rightarrow & = E \mid \varepsilon \\ M & \rightarrow & \texttt{+}\ E\ M \mid \texttt{-}\ E\ M \mid \varepsilon \\ E & \rightarrow & \texttt{id} \mid \texttt{int} \end{array}$$

Input

`id = id - id + int`

valid[0]	valid[1]	valid[2]	valid[3]	valid[4]	valid[5]	valid[6]	valid[7]

5 General (5 points)

1. Give two additional use cases for regular expressions (outside of lexical analysis).

2. What do you like about this class so far? What would you like me to keep doing?

3. What do you dislike about this class? Is there anything you would suggest changing?

4. Tell me one thing that is on your bucket list (something you would like to do before you die) or one obscure hobby you have.