CSCI-400 Exam 1

Setting up the Repository

- **READ THIS FIRST!!**
- **DO NOT CLONE THIS REPO!!!**
- This README is contained in a read-only instructor repository – you will be able to pull from here, but not push.
- To get started on the exam, first go to your *personal* repository for the exam, and carefully follow the `INSTRUCTIONS.md` file there.
- The basic idea is: you will first clone your personal repository (**not** this instructor repo), and then add a (read only) remote to pull the exam README from this instructor repository.

Instructions

- This is a take-home exam.
- There will be no lecture on the exam date.
- Just like an in-class exam, the Midterm is designed to take a maximum of 75 minutes (one class period) to complete.
- The exam will be posted to GitHub by 8:00am Mountain Time on the exam date.
- You will have 24 hours to complete the exam – the deadline for submission is 8:00am Mountain Time on the day following the exam date.
- The exam is open-book, open-notes. You may use an internet search engine, but you should *not* find a need for this, and doing so may hurt your exam performance (the internet abounds with wrong answers to a wide variety of questions).
- You should insert your answers into the indicated places in the exam text file, commit, and push to GitHub. You can push as many times as needed, up until the deadline. We will gather your completed exam for grading by cloning your personal repository after the deadline.
- Do not add any additional files. Do not add any content except for text in the exam file. Do not remove any of the existing content in the text file.
- You are responsible for ensuring that your completed exam has been pushed to GitHub properly! You can easily do this by logging into GitHub and confirming that you see the correct file contents there.

Honor Pledge

You must type your name below, to indicate that you agree to the following statements. Failure to agree to the following conditions will equate to a grade of zero on the exam. 1. "I will not discuss this exam with anyone else until after the submission deadline, and I will not obtain/share answers from/with anyone else." 2. "I understand that all submitted exams will be scanned by plagiarism detection software, just like other class assignments." 3. "I understand that if I am found to have plagiarised any content on this exam, I will receive the maximum possible penalty available to the instructors."

Type your name: Aidan Brookes

Part 1 – Functional Programming (17 points)

- **Question 1**: What are three key differences between functional programming and imperative programming?
- **Answer**:

```
In functional programming:
1. Recursion is used in place of looping.
2. Functions are first-class and can be passed to other functions.
3. Pattern matching is a core component.
```

- **Question 2**: Describe one key advantage, and one key disadvantage of functional programming, versus imperative programming.
- **Answer**:

```
Advantage: Easier testing and debugging; because functions are all self contained and h
Disadvantage: Stateless; functional programming has no notion of state. It is often use
```

- **Question 3**: When writing a functional program (say, in OCaml), how does one ensure that no stack overflow will occur?
- **Answer**:

```
Tail recursion and tail call optimization are used for this purpose.
```

- **Question 4**: Describe how to do list reversal in a functional style. Do not simply paste code – you must describe each step of your approach.
- **Answer**:

```
1. Start an accumulator list (acc)
2. Return acc if the input is an empty list
3. Split the input into the first element (hd) and a list with the rest of the elements
4. Add hd to acc
5. Return to step 1 with the new input tl
```

Part 2 – The OCaml Programming Language (17 points)

- **Question 5**: Describe what the following OCaml expression means: `(f g)`.
- **Answer**:

```
Call function f with argument g.
```

- **Question 6**: Consider the following OCaml expression, where `...` represents an unknown expression: `let f = ... in let g = 123 in (f g)+1`. What is the *type* of `f`? Also, what is the *type* of `(f g)`? Explain your answers.

- **Answer**:

  ```
  f is a function. We can assume that the ellipses contain a function definition since f
  ```

- **Question 7**: Recall that in the assignments, we have written function definitions using the form: `fun x -> e`. Using this single-argument function format, how can we pass multiple items into the function?

- **Answer**:

  ```
  Place those arguments in a tuple ie. fun (x,y) -> e.
  ```

- **Question 8**: How does OCaml's approach to typechecking differ from that of scripting languages such as JavaScript or Python?

- **Answer**:

  ```
  First, OCaml uses static typing. This means that types are checked during compile time.
  ```

Part 3 – Algebraic Data Types (17 points)

- **Question 9**: What are three use-cases for algebraic data types (ADTs)?

- **Answer**:

  ```
  Mapping (lists, trees), reducing (lists, trees), Searching within a binary search tree.
  ```

- **Question 10**: What is one major advantage of OCaml ADTs versus *classes* in Java or C++?

- **Answer**:

  ```
  The contents of an ADT can be deconstructed while maintaining type safety. For exmpale,
  ```

- **Question 11**: Recall the binary tree ADT seen in class. In an expression such as `Node(a, data, b)`, how does our program know that `a` is the *left* subtree, and `b` is the *right* subtree?

- **Answer**:

  ```
  The order of the data in the ADT is maintained, so the left subtree will always come fi
  ```

- **Question 12**: For the above expression `Node(a, data, b)`, describe how to *extract* the data from this node using OCaml.

- **Answer**:

  ```
  match node with
  | Node(a, data, b) -> (* do something with a, data, b *)
  ```

Part 4 – Syntax of Programming Languages (17 points)

- **Question 13**: What does *syntax* mean in the context of programming languages?
- **Answer**:

  `The expected structure of expressions and tokens that the compiler or interpreter will`

- **Question 14**: Context-free grammars are used for describing languages. Can they be used to describe *any* language? Why or why not?
- **Answer**:

  `No, a context free grammer can only describe langugages where the rules for production`

- **Question 15**: Consider the following grammar:
  - *sentence* ::= *subject verb object*
  - *subject* ::= **I** | **you** | **Joan** | **John**
  - *verb* ::= **ate** | **ran** | **typed** | **exhausted**
  - *object* ::= **food** | **email** | **marathon**

  Is the following a valid sentence with respect to the grammar? If so, describe what its *derivation* would look like. `marathon exhausted John`

- **Answer**:

  `No, it does not adhere to the syntax of "sentence."`

- **Question 16**: Is the following a valid sentence with respect to the above grammar? If so, describe what its *derivation* would look like. `I typed food`
- **Answer**:

  `Yes. sentence -> subject verb object -> I verb object -> I typed object -> I typed foo`

Part 5 – Lexing and Parsing (17 points)

- **Question 17**: What is the purpose of a lexer?
- **Answer**:

  `A lexer turns a sequence of characters (ex. code) into a sequence of tokens.`

- **Question 18**: What is the purpose of a parser?
- **Answer**:

  `A Parser turns a sequence of tokens into a tree given rules about the order of those to`

- **Question 19**: Consider the following grammar:
  - *e* ::= *n* | *uop e* | *e bop e*
  - *uop* ::= `-`

  - *bop* ::= `+` | `-` | `*` | `/`

  How many parse trees exist for the following expression, and why? `1 + 2 * 3`

- **Answer**:

  ```
  There are two trees depending on whether we observe the standard order of operations. T
  ```

- **Question 20**: What does it mean when there are more than one parse trees for some expression? Can we ensure that there is at most one parse tree for all input expressions? If so, how?

- **Answer**:

  ```
  Multiple possible parse trees mean that the grammer is ambiguous. It can only be fixed
  ```

Part 6 – Big-Step Operational Semantics (15 points)

- **Question 21**: What does *semantics* mean in the context of programming languages?

- **Answer**:

  ```
  Semantics in programming languages is about what code is doing on the machine after it
  ```

- **Question 22**: What is the purpose of big-step operational semantics?

- **Answer**:

  ```
  Big-step operational semantics is used to translate the entire functionality of a progr
  ```

- **Question 23**: According to our JavaScript operational semantics, what value is produced when we evaluate the expression `! ! 42`, and why?

- **Answer**:

  ```
  true
  ```

- **Question 24**: Describe how to use the operational semantics rules to show that `! ! 42` evaluates to the value you have written above.

- **Answer**:

  ```
  1. Use the Val rule to evaluate 42. This will give you 42.
  2. Use the Not rule to evaluate !v where v is 42. This will give us the mathematical ex
  3. Use the Not rule to evaluate !b where b is false. This will give us the mathematical
  ```