COMP 333 Final Practice Exam

The final exam is cumulative. This practice exam, **in addition to** the prior practice exams, assignments, in-class handouts, and exams, is intended to be a comprehensive guide for studying. This practice exam only focuses on material since the last exam. You are permitted to bring three 8.5 x 11 sheets of paper into the exam with you, as long as they have handwritten notes on them. Both sides of both sheets can be used. To be clear, these must be entirely handwritten.

Language Terminology / General Concepts

1.) Java and JavaScript both use garbage collection for memory management. In contrast, Rust uses ownership and borrowing.

1.a.) In 1-3 sentences, in your own words, explain how garbage collection reclaims memory. Your description doesn't have to be detailed enough to implement a garbage collector, only detailed enough to get the gist of when memory would be reclaimed.

Starting from all variables declared on the stack (or globally), we trace through memory and determine which parts of memory are reachable or not. The parts of memory which are not reachable are reclaimed. This tracing and deallocation is performed at runtime.

1.b.) In 1-3 sentences, in your own words, explain how Rust's ownership and borrowing system reclaims memory. It doesn't need to be detailed enough to implement it, only detailed enough to know when memory would be reclaimed.

Every bit of allocated memory has a single unique owner, which are traced all the way back to variables declared on the stack. When a stack variable is deallocated, any memory owned by the variable is deallocated. The deallocation itself is performed at runtime, but we can determine when these deallocations happen at compile time.

1.c.) Name one advantage of garbage collection over Rust's ownership/borrowing system.

Possible answers, among others:

- Garbage collection is fully automatic, and you generally don't need to think about it
- Can have many references to the same object without restrictions (e.g., you don't have to designate any of them as the owner, nor can you)
- Overall more flexible
- More popular - better understood by more people

1.d.) Name one advantage of Rust's ownership/borrowing system over garbage collection.

Possible answers, among others:

- Memory is reclaimed as soon as it is no longer needed
    - Likely still in a cache, so reclamation will likely be faster
    - Less wasted memory overall
- Predictable behavior at runtime, making it appropriate for real-time systems
- No runtime component for memory management - less overhead

2.) In 1-3 sentences, explain the difference between compilation and interpretation. Your answer does not need to be detailed enough to implement a compiler or interpreter.

Compilers translate programs written in one programming language to another, whereas interpreters directly execute programs written in a given programming language.

3.) The Java Virtual Machine (JVM) is implemented as an interpreter over Java bytecode. Similarly, most JavaScript implementations are implemented as interpreters. However, most Java and JavaScript implementations support just-in-time (JIT) compilation.

3.a.) In 1-3 sentences, explain what JIT compilation does, in the context of an interpreter. Your answer doesn't need to be detailed enough to implement a JIT compiler.

The interpreter dynamically monitors code and can choose to compile chunks of frequently-executed code directly to machine code. When these chunks are executed in the future, it can jump to the compiled chunks of machine code as opposed to interpreting the corresponding original code.

3.b.) JIT compilers can sometimes generate faster code than traditional compilers. Why?

Traditional compilers can only reason about how code *might* execute, whereas JIT compilers can observe exactly how code *is* executed. As such, the JIT compiler has more information to work with, and can use this to generate faster, more optimal code.

4.) C only has support for first-order functions, whereas JavaScript has support for higher-order functions.

4.a.) In 1-3 sentences, explain what higher-order functions are. You don't have to provide enough detail to explain how to use them.

Higher-order functions allow for functions to be created dynamically at runtime. As part of this, functions become another kind of data, so they can be saved in variables, passed to other (possibly higher-order) functions, and returned from (possibly higher-order) functions.

4.b.) Unlike first-order functions, higher-order functions may require memory to be dynamically allocated at runtime. Why?

They might "close-over" a value from an enclosing scope. For example, consider the following JavaScript snippet:

```
function foo(a) {
  return function (b) {
    return a === b;
  }
}
```

The function returned by `foo` needs to save `a` somewhere, and `foo` could be called an arbitrary number of times. As such, dynamic memory allocation is necessary.

4.c.) Write a JavaScript code snippet that uses higher-order functions and would require memory to be dynamically allocated at runtime.

The snippet from 5.b. is an example of one such function.

5.) Consider the following code snippet, which is written in some unknown programming language:

```
DefineFunction foo(x, y):
  DefineVariable temp = x dividedBy y
  return temp

foo(3, 4)             // first call to foo
foo("alpha", "beta") // second call to foo
```

5.a.) Assume this language is statically-typed. Does this code probably compile? Why or why not?

Probably not. foo seems to be doing division, but it's called with both integers and strings. Division doesn't make sense for strings.

5.b.) Assume this language is dynamically-typed. Does this code probably compile? Why or why not?

It probably compiles, even though there will likely be a runtime error at the second call to `foo`.

6.) Consider the following Java function:

```
public static void foo() {
  int x = 0;
  Object obj1 = new Object();
  Object obj2 = obj1;
}
```

6.a.) If `foo` is called, what is allocated on the stack?

`x`, `obj1`, `obj2`

6.b.) If `foo` is called, what is allocated on the heap?

a single `Object`

6.c.) When `foo` returns, what is guaranteed to be deallocated immediately?

Anything allocated on the stack, namely `x`, `obj1`, and `obj2`

6.d.) When `foo` returns, is there anything which might be deallocated at a later point? That is, what is *not* guaranteed to be immediately deallocated?

Anything allocated on the heap, namely the single `Object`

Rust

7.) Declare a struct named Example that holds five fields:

- `first`, which holds a `String`
- `second`, which holds a 32-bit unsigned integer
- `third`, which holds a 32-bit signed integer
- `fourth`, which holds a 64-bit unsigned integer
- `fifth`, which holds a 64-bit signed integer

```
struct Example {
  first: String,
  second: u32,
  third: i32,
  fourth: u64,
  fifth: i64
}
```

8.) Consider the following Rust code:

```
fn main() {
    let s1: String = "foo".to_string();
    let s2: String = s1;
    println!("{}", s1);
}
```

This code does not compile, and the compiler produces an error message pointing to the use of `s1` in the `println!` statement. Why doesn't this code compile?

Ownership of the `String` was transferred from `s1` to `s2`, so `s1` can no longer access the `String`. However, this program still attempted to access the `String` through `s1`.