COMP 333
Summer 2025
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.

JavaScript

1.) Consider the JavaScript code below and corresponding output, adapted from the second assignment:

```
function Cons(head, tail) {
  this.head = head;
  this.tail = tail;
}
function Nil() {}

let list = new Cons(1, new Cons(2, new Cons(3, new Nil())));
list.forEach((x) => console.log(x));

---OUTPUT---
1
2
3
```

Implement any missing code necessary to produce the above output.

Language Concepts

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.

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.

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

Memory Management

4.) Java is a garbage-collected language. Consider the following Java code.

```
public static void foo(int x) {
  int a = 3;
  Object b = new Object();
  int c = 4;
  Object d = b;
  Object e = new Object();
}
```

Assume that `foo` is called.

4.a.) What will be allocated on the stack over the duration of `foo`'s call?

4.b.) What will be allocated on the heap over the duration of `foo`'s call?

4.c.) When will the stack-allocated components be deallocated?

4.e.) When will the heap-allocated components be deallocated?

5.) Consider the following Java code:

```
public class Foo {
  public int x;
  public Foo(int y) {
    x = y;
  }

  public static void main(String[] args) {
    Foo obj = new Foo(1);
  }
}
```

For each variable in the program, list whether the variable will be allocated on the stack or the heap.

6.) C requires users to explicitly allocate and deallocate heap memory. This is in contrast to automated memory management techniques, wherein deallocation is performed automatically.

6.a.) Name one advantage of explicit deallocation over automated deallocation.

6.b.) Name one advantage of automated deallocation over explicit deallocation.

7.) What is wrong with the following C code performing explicit memory management?

```
void foo() {
  int* p = malloc(sizeof(int));
  *p = 5;
}
```

8.) What is wrong with the following C code performing explicit memory management?

```
void foo() {
  int* p = malloc(sizeof(int));
  *p = 5;
  free(p);
  *p = 6;
}
```

9.) What is wrong with the following C code performing explicit memory management?

```
void foo() {
  int* p1 = malloc(sizeof(int));
  int* p2 = p1;
  free(p1);
  free(p2);
}
```

10.) What is wrong with the following C code performing explicit memory management?

```
int* foo() {
  int* p1 = malloc(sizeof(int));
  free(p1);
  return p1;
}
```

11.) Programs generally can dynamically allocate memory in one of two places: the stack and the heap.

11.a.) Name one advantage of stack allocation over heap allocation.

11.b.) Name one advantage of heap allocation over stack allocation.

12.) There exist multiple automated memory management techniques. For example, Java and JavaScript both use garbage collection, Python and Swift use reference counting, and Rust uses ownership and borrowing.

12.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.

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

12.c.) 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.

12.d.) Name one advantage of reference counting over garbage collection.

12.e.) Name one advantage of garbage collection over reference counting.

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

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

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

13.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.

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

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

14.) Consider the following Java function:

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

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

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

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

14.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?

Rust

15.) 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

16.) 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?