Computer Science 245
Test 1 review.
Test 1: Friday, Feb 20, 2026

This test will exercise your progress in programming under the following headings:

C programming.

- Simple ways that C differs from Java: no boolean type (instead, use `int`), no String type (instead, use `char` arrays terminated by `\0`), statically allocated arrays.
- Header files and implementation files; recall that header files are for function prototypes (and defining struct types) and implementation files are for the implementation of functions.
- Struct types and struct values; fields; how to define a struct type and use a struct value.

Invariants and algorithmic analysis.

- Loop invariants as propostions that capture a relationship among the variables that does not change as the loop progresses.
- The line-by-line method of analyzing an algorithm's efficiency; the (best- or worst-) case running time of an algorithm as a function of the size of the input; big-oh categories, expecially $O(1), O(\lg n), O(n), O(n \lg n)$, and $O(n^2)$.

Java and OO concepts and vocabulary.

- Basic Java and OO vocabulary:

- Object
- Class
- Member
- Instance variable
- Instance methods
- Instance of a class
- Composite type
- Reference type
- Method
- Receiver
- Constructor
- Overloading
- Static method
- Static variable
- Static initializer
- Method signature
- State class vs value class
- Interface (concept)
- Subtype, supertype
- Polymorphism
- Static type vs dynamic type

- Java language features: `this`, access modifiers (`public`, `private`), `iterface` (construct).
- The difference among *kinds* of variables: Local variable, instance variable, static variable, formal parameter.
- Kinds of members of a class: instance variable, instance method, constructor, static variable, static method, static initializer.

Linked lists.

- Concepts and terms: Node (object), node class, head reference, link.
- Algorithmic concepts: Pattern for a loop over the nodes in a class; adding to the front of the linked list, removing a node.

ADTs and Java Collections.

- The concept and definition of an abstract data type; ADT examples: List, Map, Set.
- Interfaces: `List`, `Map`, `Set`, `Comparable`, `Iterator`.
- Classes: `ArrayList`, `HashMap`, `HashSet`.
- (I will provide the names and signatures of relevant methods for these classes and interfaces, but you need to know what they mean and how to use them.)

Kinds of problems to expect.

- Short-answer questions about basic C programming.
- Demonstrate vocabulary by finding examples of various things in a given piece of code (such as, "give an example of a static variable."
- Given an expression in context, determine its static type and dynamic type.
- Given a short algorithm, write a loop invariant, determine a running time from a line-by-line analysis, and give a big-oh category.
- Write a portion of code in C to operate on a string represented as a char array.
- Write a portion of code in C to define a struct type and functions to operate on that type.
- Write a portion of code in Java to finish a class or method.
- Write a portion of code in Java to implement a linked list operation.
- Write a portion of code in Java that makes use of Java collections classes.

Practice problems.

1. a. Assuming a `Node` class as we saw in class, complete the following method that takes the head node of a list and computes and returns a list like the given one except that the nodes with odd-numbered data are removed. The remaining data in the new list should be *in the same order* as the original list. Moreover, do not mess up the old list; create a new one.

```
/**
 * Make a new list from the given one containing only the values in even-numbered positions in the original
 * (indexing from 0); the given list is to be undisturbed. The new list will, accordingly, contain all new nodes.
 * @param original The list whose evens we want
 * @return A list containing only the values in the even-indexed positions in the original
 */
public static Node evensList(Node original) {
```

b. Analyze the running time of your algorithm from part a.

2. Suppose you are working on an application in C that needs to model rational numbers (i.e., fractions). In this problem you will make a new C type `rational` by writing a struct and operations to go along with it.

The `rational` type should have operations `fraction` (taking two `int`s and returning a new `rational` value), `add` (taking two `rational`s and returning the sum), `mul` (taking two `rational`s and returning the product), `reciprocal` (taking one `rational` and returning its reciprocal) and `display` (taking one `rational` and displaying it on the screen, showing something like `3 / 5`).

(For an extra challenge, make sure that the fractions returned by these functions are in simplest form; you may assume you have a function `gcd` that computes the greatest common divisor of two `int`s.)

Write a header file `rational.h` and an implementation file `rational.c` so that the program on the next page would work.

```
#include<rational.h>

int main()
{
    rational a, b, c;
    a = fraction(1, 2);
    b = fraction(2, 3);
    c = add(a, b);
    display(c);

    c = reciprocal(mul(a, b));
    display(c);

    return 0;
}
```