WASHINGTON STATE DEPARTMENT OF
NATURAL RESOURCES

October 23, 2023

Dear candidate,

The hiring committee has reviewed the code samples that you submitted to us. We were pleased to note that these samples demonstrate a good level of ability and commitment to the craft of software development, and we are interested in continuing to get to know you and what you would bring to the Forest Data Systems Manager role.

As one step in the screening process, we would like to ask you to complete a couple of programming exercises. This document describes the requirements for these exercises and explains how to submit your work.

If for any reason you are not able to complete the exercises in the time allotted, please submit whatever work you have done, along with a brief explanation of where you left things and what remains to be completed.

You may consider these exercises an "open book exam," and so make use of online reference materials such as you would have available to you if this were an actual on-the-job assignment. You are also welcome to reach out to the development team here in the Forest Informatics section at DNR if you have any questions about the goal of the exercises or need clarification about the requirements. Please direct any such questions to Erin Crosland (erin.crosland@dnr.wa.gov).

In working these exercises, please favor clean and expressive code over (possibly premature) performance optimizations. You may write your solutions in any commonly available programming language.

Exercise 1. Classify forest inventory units

Along with these instructions, you should have received a comma-separated values (CSV) file called *trees.csv*. This file contains a simplified *tree list* made up of randomly generated data. The goal of this exercise is to write a program that classifies each *forest inventory unit* represented in the dataset as either Class A, Class B, Class C, or Class D, according to the criteria laid out below. (This is a hypothetical classification invented for the sake of this exercise.)

Each record of the input dataset represents a tree belonging to some forest inventory unit. Each record contains three fields, as follows:

- *unit_id*: the unique identifier (UUID) for the forest inventory unit to which the tree belongs; you may assume that the dataset is sorted such that all records for a given unit occur consecutively;
- *species*: a standardized two-letter code indicating the tree's species (e.g., DF=Douglas-fir);
- *dbh*: standing for diameter at breast height, this is a standard measurement of the tree's diameter taken 4.5ft above ground level; as expressed in this dataset, this is a decimal number in units of inches having a single digit of precision.

You may assume that the input dataset is clean, and so omit validation checks from your code.

For the purpose of our classification, let us define two groups of tree species that are of interest to us. We call these groups *conifers* and *hardwoods* respectively. The input dataset may contain trees of species not listed here, but such trees must be ignored by the classifier.

> conifers = {DF, WH, AF, ES, LP, WP}
> hardwoods = {RA, WA, AS}

The classification which your program must observe is laid out in the following table. Note that these classes are ordered, with classes towards the top of the table being preferred to those further down. Therefore, if a forest inventory unit meets the criteria for both Class A and Class B, for example, then your program must report it as being Class A.

> Class A:
> at least 10 conifers with DBH greater than or equal to 20"
> *and* at least 10 additional conifers with DBH greater than or equal to 12"
> Class B:
> at least 20 conifers with DBH greater than or equal to 12"
> Class C:
> at least 10 hardwoods of any diameter
> Class D:
> catch-all for any inventory unit not otherwise classified

Your task:

Please write a function (method, subroutine, etc.) that accepts a list of tree records for a single forest inventory unit and returns a string representing the class to which the unit belongs according to the classification scheme laid out for this exercise.

You may choose how you represent the tree records within your program. You may, if you wish, define other helper functions or datatypes also. Finally, to pull everything together, create an entry-point to the program (i.e., a main function) that accomplishes the following tasks:

1. read and parse the input dataset in *trees.csv*, arriving at a list of tree records per forest inventory unit;
2. pass each list of tree records to your classification function in turn, retrieving the class for the unit under consideration;
3. write an output file called *classification.csv* that contains one record per forest inventory unit, with each record comprising the fields, *unit_id* and *class*.

Exercise 2. Classify forest inventory units using configurable business rules

As time allows, after completing Exercise 1, please give some thought to this follow-up exercise.

The goal of this exercise is the same as that for Exercise 1, namely to classify a collection of forest inventory units. Where this exercise differs from the former exercise is that here we add the requirement that the criteria according to which an inventory unit is classified be configurable by users outside of our development team.

In production, perhaps we will load these criteria from a database or receive them as part of a request to a web API. Regardless, your task for today is to generalize the classification function you wrote for Exercise 1 so that it accepts, in addition to a list of tree records, specifications for the classification rules.

For the sake of this exercise, assume that the allowable classification schemes follow the pattern we established with the previous exercise. More formally:

- a classification comprises a list of zero or more classes;
- each class has a name and a set of zero or more criteria;
- a class is satisfied when all of its criteria are satisfied (i.e., we use logical AND);
- the first class to be satisfied is returned as the outcome of the classification;
- if no class is satisfied, the classification function returns the special string, “Unknown”.
- each criterion has the general form: *N* or more trees where species is in a well-known set *S* and DBH is greater than or equal to *D*;
- for our purposes, *S* is either conifers or hardwoods, as these were defined for Exercise 1, *N* is a non-negative integer, and *D* is a non-negative real number.

Your task:

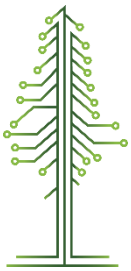
Design a way to represent a classification scheme within your program. Then create a classification function that accepts a classification scheme and a list of tree records for a single forest inventory unit. As for Exercise 1, the function must return a string representing the name of the class to which it assigns the inventory unit, although in this case it may also be the special string, “Unknown”.

Now express the classification scheme laid out for Exercise 1 in terms of the representation that you designed here. Create an entry-point to your program similar to the one you created for Exercise 1, but pass your representation of the classification scheme to the classification function along with the list of tree records. (Alternatively, if using object-oriented programming, perhaps the classification scheme would be stored in an object instance variable instead.)

Submitting your work

This section describes how to submit your work on these exercises for review by the hiring committee.

1. Please create a Git repository to contain your work, and grant us access to it. You may either submit a link directly to the repository on GitHub or a similar site, or go through GitFront. We must be able to *git clone* your repository.
2. Create two directories within the repository, one for each exercise. Call these *exercise1* and *exercise2*. (If you prefer to factor out code that’s shared between your two solutions, you may do so, and alter the organization suggested here. Just be sure it’s clear which file contains the entry-point to your solution to Exercise 1 and which contains the entry-point to Exercise 2.)
3. Each exercise directory should contain the following files:
    a. one or more source files implementing your solution to the exercise;
    b. a CSV file called *classification.csv* that contains the results of running your solution against the input dataset contained in *trees.csv*.
4. Create a readme file at the top level of the repository that briefly explains how to compile each of your solutions (if necessary) and run them. You may assume that the reviewer has the basic language tools available—e.g., a Python interpreter or a Java JDK. Additionally, if your solution depends on a third-party library (e.g., to read a CSV file), then list the libraries that you use and tell the reviewer where to acquire them. All such libraries must be freely available.
5. Finally, reply to the email in which you were asked to complete these exercises and let us know that you’re done. Include the URL to the Git repository containing your solutions.



Thank you for your time and the effort you invest in completing these exercises.

Sincerely,
The DNR Forest Informatics Team