**Comparing Imperative and Object-Oriented Programming**

*Name: Stephen Griffin*

*ID: 18482934*

I chose to do the Imperative vs Object-Oriented comparison using C as my imperative language and Python3 as my object-oriented language. Imperative programming is a paradigm that describes the steps of how the computer should accomplish tasks instead of what it should accomplish. Object-oriented programming is the concept of using objects that hold code and data.

My object-oriented approach would require a bit more thought than the imperative as I would need to use objects and classes. For my object-oriented approach in python I created a class that would act as the nodes in the binary tree, here I initialised my variables which included the contact details as well as the parent, left node and right node. I then created class for the binary tree itself which would act as an object containing the functions for inserting, printing, searching and deleting.

A class is like a template that outlines fields with methods that act on fields. They are useful because we can create instances of these classes which are called objects. Doing this allows us to make use of public/private fields. This is referred to as encapsulation and it makes the code a lot easier to debug because you know where to look rather than being faced with a massive chunk of code. Due to the specification requiring two trees; one for insertion/searching by name and another for insertion/searching by number I would need two of these binary tree class objects.

For my imperative approach I defined a function to allocate memory to a node, assign the phonebook data structure and return it. Imperative programming is seen as a linear approach because it doesn’t need a lot of planning. This is because we are dealing with a single state. Next was an insertion function for adding nodes into binary tree 1 based on name. If the tree was empty a new node was made as the root, I checked if the new name was greater than the roots name, if so then it would be inserted right, if it was less it would be inserted left. This was more or less repeated for binary tree 2 except it was to search by number. This method of insertion carried over to my python implementation where I took the same approach to insertion.

To search for a name in the first binary tree of my python implementation I would check whether the name being searched for matched the current nodes name, if so it would return that node, otherwise it would check the left node for the name, then the right, continuously until it finds the name or it is not found. In my python implementation I used some features of imperative programming in parts, such as direct assignment and state change, but usually this was combined with encapsulation which is an object-oriented feature.

My imperative implementation included a header file which contained my declarations of the phonebook structure, the node structure, and following functions. My main.c file included a driver function which contained a simple interface to be used in terminal for the insertion, deletion and printing of data. I also incorporated this simple terminal interface into my object oriented python approach.