The imperative is written in C and the object oriented part is written in C++. I had originally chosen to use Java, but switched to C++ when I found that Java did not support operator overloading. This was a mistake. I greatly overestimated my own understanding of the language and lost two days to strange syntax errors that google couldn't help me solve. When all is said and done, though, there is hardly a better pair of candidates for such a comparison than C and it's object-oriented offshoot and, having now seen the difference, I will concede that C++ isn't that bad.

For the imperative part, it was originally my plan to make all functions take no parameters and return nothing, so that every function would only alter the global state. I thought this would help differentiate it from functional programming as much as possible however it ended up being far too difficult to be worthwhile (not to mention it would be truly horrific to read). The program still has some global state and plenty of void functions but it would ultimately take little effort to convert it to a functional style.

At the top of phonebook.c, there are two type definitions followed by their associated constructors and destructors, here is the first place where classes would make an improvement: These functions have a tight coupling with the associated type and it makes sense that they should be, in some way, part of the type. In the Object oriented program, these become the constructor and destructor class methods, which makes things a little bit clearer and more organised. The display functions are also well suited to class methods for the same reasons as above.

The main benefit I got from object oriented programming was an easy way to deal with the different types of Number and Name. The program has two different types of node; one keyed by name, the other keyed by number. In the object oriented program I was able to encapsulate this difference with just one line in each child node. In the imperative program I still had options; macros, function pointers, or just duplicating the code, but it would be far harder to make the resultant code clean and readable. As such, I've decided to split the imperative version of the program into two versions; one which just uses integer keys, and another which uses number and name.

There are three main reasons why C++ served this need better than C: The first is templates, which allow somewhat type-agnostic code to be written so long as the type is eventually specified, this is adjacent to OOP but not necessarily a result of the style. The second is the use of the string class, instead of C's more primitive char pointer, which allows me to compare strings the same way I compare ints, this is a result of OOP, but could have been used even if my specific program was not written in that style. The third is the use of inheritance which allowed me to change the specifics of how functions operate based on the type, and not need the caller of those functions to know the difference.

Overall, I think that this particular program is very well suited to OOP, as is generally the case for something which could be thought of as a type (like a linked list or a BST). I kind of wish that you could have picked something where the superior style was less clear.