The due date on this assignment will be left open for the moment; we’ll see how progress on it goes.

Develop a data structure for the language of first-order logic with the usual propositional connectives and quantifiers, a countable supply of variables, and a countable supply of predicates and functions of each arity. You can make your own decisions about the allowed shapes of variables and whether functions need declarations (to handle the arity for example) or might have syntactical features that tell one what the arity is.

You need a parser and display functions for this data structure. You have design freedom on the exact syntax you use (and on questions like order of operations though I suggest that making parentheses always mandatory works against usefulness).

The second part of this assignment is to write a substitution function \( \phi[t/x] \) which will handle the result of replacing any variable \( x \) with any term \( t \) in any formula \( \phi \) (notice that you will need the same thing for terms, since having functions of all arities and variables we can build complex terms). The main point of this is to avoid the disasters which can occur if you make substitutions carelessly in the presence of bound variables. So you need to recognize free and bound occurrences of variables and recognize situations where bound variables need to be renamed. I suggest a conservative strategy rather than the aggressive one of renaming all bound variables found in Marcel (check whether renaming is actually needed).

I would also like you to write a function which will check whether two formulas or two terms are the same up to renaming of bound variables.

You might notice that such a function is a prerequisite for correct implementation of quantifier rules.
If you want to support the syntax of a type theory (have types, hidden or visible) on your variables that occasions other design choices. There is no requirement that you do type theory at this stage, but you might want to think about how you would add it (and how you would type check terms, and what extra declaration information functions would need, and so on).

If you want to support infix notation for binary functions and predicates be my guest.

Code should be commented locally and there should be separate documentation explaining the technical problems with defining substitution. All code should be your own, of course. You may use general purpose parser building tools if you know how to do this.