This language can represent two different kinds of types: bool and int. Using expressions, this language can perform computations, ranging from addition to integer division. Aside from that, this language can also compare different expressions using the common operands such as greater than and less than, equal to, not equal to, and many of the other common ones seen in mathematics. Through the visitor pattern that was implemented, users can perform operations on the expressions that were defined by the abstract syntax tree. Along with the AST, the evaluate function was also implemented along with type verifying and printing.

The base Type class in "ast.hpp" defined at the beginning has a visitor struct "Visitor" and a member function which accepts an object by reference. Bool_type and Int_type inherit from the "Type" structure. The Expr::Visitor defines all of the virtual visitor functions. All of those functions can take different type of arguments all based on the different expressions that were defined within the language. Everything was written in C++ as requested by my professor. All the classes that define the abstract syntax tree are within "ast.hpp".

The functions defined in "evaluate.hpp" let you evaluate an expression by taking an expression and returning a value. The type verifying function makes sure that values being passed are of the right type before being created. The function will check to see if the value being passed is either a "bool" or "int", according to the rules given by the professor. The print function is nothing more than a print function. It simply prints your output. I also included a function that checks for parentheses. I figured this function would be useful because it outputs the expression in a nice format.

Source code: https://github.com/draganjovic/Compiler-Design