Translator Design - Lab 2

1. Consider the exponentiation function from `examples/func.imp`. Unfortunately, variable declarations are not yet supported. Rewrite `exp` without the use of variables, ensuring the time complexity is unchanged.
2. Implement the missing operators:
   (a) Add the required tokens to the lexer.
   (b) Define the appropriate binary operators.
   (c) Produce the AST nodes in the parser, taking care of precedence and associativity.
   (d) Define the instructions implementing the operators.
   (e) Translate the AST nodes to instructions.
3. The function now relies on `if` statements - add the keywords used in if statements to the lexer.
4. Define an AST node to represent `if` statements, ensuring the presence of an optional alternative branch.
5. Compile `if` statements to bytecode.
   (a) Identify the number of labels required and allocate them.
   (b) Lower the condition check and emit a conditional jump.
   (c) Recursively compile the bodies of the branches.
   (d) How would the `if` statement be lowered if there was a `JUMP_TRUE` opcode?