1 Error Detection in WLP4

1. There is no `^` token in WLP4. This is a lexical error.
2. The variable ‘y’ is declared twice. This is a semantic error.
3. The variable ‘x’ is divided by 0. This is a run-time error.
4. The variable ‘idx’ is not declared at all. This is a semantic error.
5. In WLP4, all variable declarations must proceed all statements. This is a syntax error, since the WLP4 grammar forces this structure.
6. We don’t have tokens for character literals (in single quotes) in WLP4. This is a lexical error.
7. We have a duplicate definition of sub. Although they have different number of parameters, function overloading is not a feature in WLP4. This is a semantic error.
8. Function f is declared before function g, therefore f has no knowledge about g’s existence. Note that this prevents the use of mutual recursion in WLP4. This is a semantic error.
9. This program runs indefinitely. This is a run-time error.

2 Symbol Table Error Checking

In the assembler, we needed to do two passes because labels could be used before they were declared. In WLP4, we require declaration before use. That is, we know immediately that it is an error to see an unknown identifier in any given variable use or function application. In fact, the WLP4 grammar takes care of the former for us!

3 Code Generation

1. 

   statement → FOR LPAREN aexpr SEMI test SEMI aexpr RPAREN LBRACE statements RBRACE

2. for needs to be added as a keyword.
3. Our new rule is well-typed iff both aexprs, the test, and the statements are all well-typed.
4. `code(statement FOR LPAREN aexpr1 SEMI test SEMI aexpr2 RPAREN LBRACE statements RBRACE) = {
    code(aexpr1)
    startLb = genLabel()
    endLb = genLabel()

    startLb:
    
    code(test)
    ; If the test failed, skip the loop
    beq $3, $0, endLb

    code(statements)
    code(aexpr2)
    beq $0, $0, startLb

    endLb:
    }
`