1. Semantic Analysis

First we start by saving some of the information gained during semantic analysis in the AST objects, which we will need in later phases of the compiler. All of these tasks should be fairly trivial as all the information is already around in the visitor which we built during the last week.

- Type of the expression in the `WriteStatement`. We will need to know which exact function to call, without being able to rely on overloading
- Type of the return-value of an `ArrayAccess`. We will need to know if we get another address of an array back; or if the value retrieved can be interpreted as an integer.
- Check if the `int Tiny()` function is around.
- Purge the list of functions so that only actually referenced functions are there.
- Purge the list of local variables for the same reason.

2. Converting AST to Three Address Code

In the second step to transform our high-level AST to binary code, we will now transform our high-level AST "graph" to the lower-level three address code format. This means that we will have to serialize all nested structures (`a = 3*x+b/c;` becomes `temp1 = 3*x; temp2 = b/c; a = temp1+temp2`). Secondly we have to replace control structures like `while` and `if..else..` in combinations of labels, tests and jumps. Here you have to pay attention that nested control structures are expanded correctly.

You will be able to find new tests for `Ast2Tac`, the three address code classes, stubcode and the updated AST in the SVN repository:

`https://www.iam.unibe.ch/scg/svn_repos/Lectures/CompilerConstruction/TinyExercises/`