A5 Testing Design

Overview

In our approach to testing for this assignment, we decided to reorganize all of our existing test cases from the past three assignments and amalgamate them into one cohesive testing framework. However, there were some drawbacks to this idea, mainly in the large number of tests written for A2 which were designed to pass syntax analysis but fail semantic analysis. Most of these needed to be migrated into the failing set of tests, which are organized by which assignment they were first implemented for. In contrast, the set of passing tests are organized by the source language features or constructs, as they number much greater and are more easily navigated using these categories.

Passing

As we have done in the past, we opted for a large number short, small test cases intermixed with a few choice longer tests for our testing strategy. The long tests were carried over from our very first assignment, while the shorter tests are a mixture of tests written for A3, rewritten tests from A2 and new tests written to cover cases that we had previous neglected. For instance, we noticed that we were quite lacking in tests for yields in terms of the different combinations of statements and expressions that could appear in a yields expression.

Failing

We were initially unsure as to what tests we could write that would intentionally fail at the code generation/execution stage, since most errors that we are familiar with stem from syntax or semantic issues. As mentioned previously, some tests that were designed to pass syntax checking but fail semantic analysis were moved to a folder called “prevpassing” inside the A2 directory. Additionally, we added three tests design to fail due to runtime issues. One of them is an array access out-of-bounds error, and the other two involve trying to use an uninitialized variable.

An interesting situation arises when the latter case occurs in the context of a function call. From our understanding (and later confirmed by Prof. Wortman), when a function is invoked with some variables passed in as arguments, the compiler should try to load those variables’ values for the function’s activation record, as function arguments are passed by value in our source language. If one of those variables is not initialized when the function is called, this would be an error on the programmer’s part. It should be the programmer’s

responsibility to ensure variable values are initialized before those variables get passed to a function, and the compiler should not be accountable for detecting or recovering from such errors.