Simple-O Language Specification

(for SimpleCompilerv3)

First, it is important to remember that we are not developing a fully-featured C compiler or a compliant C compiler. This language is C-like and there are concessions along with major missing functionality (this is primarily due to the fact that we are limited on time and what can be reasonably accomplished individually within a single semester).

I call our language Simple-O.

The following represents the state of the Simple-O language at the completion of all assignments in the course. Not all of these specifications are valid until that point. This document is meant to provide the structure of the language as you develop the compiler. All supplied test cases align with these specifications.

Declarations

1. All declarations must be at the top of the function (whether there is an assignment at that time or not). These should be prior to anything else in the function.
2. Assignments of integers, unary, and binary expressions are supported. Assignments requiring more than a binary expression should use another variable and a subsequent binary expression assignment using the previous variable and an integer or other variable. If you want to assign a unary or binary expression to a variable, declare the variable first (unassigned) and assign the expression as a separate line (e.g., "int x; x = y + z;").
3. Variable declarations can include explicit assignment at the time of declaration or later (see below):

```
int myFunction(int a)
{
   int x;
   int y;
   int alpha = 500;
   int z;

   a = 75;
   x = 90;
   y = 100;
   x = 66;
   a = 5;
   z = 20;
   y = 32;

   return y + 1;
}
```

Numbers

1. Only integers are supported.
2. All integers are unsigned (inherently positive only) and 32 bits in size.

Variables

1. Variables can be up to 20 characters long (including null terminator). Variables must only contain letters (upper and lower case) and no numbers.

2. Only local variables are supported.

Storage Classes

1. Only the “auto” storage class (i.e., stack-based memory for variables) is implemented, but this cannot be explicitly listed in the source code). This is what determines that only local variables are supported (via the stack).
2. No explicit storage classes of any type are supported.

Programs and Functions

1. All functions must return a value (either zero or the intended return value of the function).
2. Only functions returning integers are supported.
3. The symbol table can only support up to 100 entries (including all types of entries you may choose to store in the symbol table during compilation beyond the variables in the input program).
4. All programs inherently only support one function. If you wanted to create a binary with two functions, you would create a Simple-O source file for each function, compile them separately using SimpleCompiler, then link them together via the OS linker into the final binary along with an appropriate driver file.
5. Function names follow the same rules as variables, but the evaluator code in the test-cases directory will expect your function to be named “myFunction” for automated testing.
6. Functions can include one integer argument or no argument. (System V ABI for x86-64 is assumed as the calling convention. No other calling conventions are supported.)

Operators

1. Addition (+), subtraction (-), multiplication (*), and logical negation (!) are supported. (Division is not supported.)
2. Less Than (<) operator is supported (only in a Do While loop).
3. Increment (++) is supported.

Control Structures

1. Do While loop is supported. (No other loops are supported.) Only one loop can be used in a function.