COMP3235 Assignment 3
Due: May 19, 2023

*Add the following two features to your c5 language, producing the c6 language and compiler.

Feature 1 [50%]. Multi-dimensional arrays.
Since the compiler needs to pre-allocate space, so arrays (unlike scalar variables) must be declared beforehand (but anywhere):

```
array foo[9,22]; // defines a 2D array with indices 0..8, 0..21
array x[3], y[10,10,10];
array z[100] = 0; // initialize all elements of z to 0
```

Assume all non-linear arrays to be stored in the row-major fashion. To keep to the non-typed character of our language, we treat "foo" etc. above as just variable names, and "[…]" as memory allocator (when in an array declaration) or an index into a memory location relative to "foo" (when used in an expression).

```
foo[3,12] = 499; x[2] = "hello world";
x = 1234; // x treated as x[0]
foo = "o my god"; // actually equivalent to foo[0,0] = ...
foo[3] = 5678; // this is the 4th element from foo; not foo[3,0]
foo[2,] = ...; // Illegal! Let's disallow all "incomplete" indices
```

Feature 2 [50%]. Functions and function calls.
A function can have zero or more formal parameters. A parameter is a single/scalar variable (integer, char, a nas string, or an array name), and arguments are passed by value. A function's name should not clash with that of a variable. Function definitions may appear anywhere in the code (although defining them all at the beginning is a good practice), and so you might need an extra pass (or backpatching) to fill in their addresses in calls. A function has a single return value. Naturally, functions can be recursive.

```
prime(x) {...} // this defines a function
a = prime(myint); // this calls a function
prime(x); // or this, if prime() does not return a value
if (func(a, b, c)) ...; // calling a function with three parameters
simple() {...}; a = 24 + simple(); // a function with zero parameters
```

If passing an array name (e.g., "foo") as argument, the first element of the array (i.e., "foo[0]") is passed instead. A function can only be declared at the outer (the main) scope, not within another function definition. A function can have its own local variables (including arrays). A function can refer to variables in the outer scope (and only those in the outer scope) if these variable names do not clash with the function's local variables. If there is a clash, the local variable is the one to be used.

Bonuses [additional 20%]. To be announced on 27.4.2023 along with Tutorial 4 (important; please try to attend).

Revised Assignment 3

Single-dimensional arrays only [40%].

No change [60%]. You may choose to use the new nas2 instead (please refer to T4).

Bonuses: (a) Multi-dim arrays [+10%].
(b) Functions can have arrays as parameters [+10%].