# CMPUT 415 - Team YACC - Compiler Documentation

Sarah Anderson
Marko Babic
Guanqui Huang
Aaron Krebs
Shihao Xu

## Further Improvements on Semantic Checking

Since checkpoint 2, we have made further improvements on semantic error checking. If a program re-declares a type, we show an error and do not perform the re-declaration. if the user re-declares a type, there are two likely scenarios for how this type will be used in the remainder of the program we are parsing. The first scenario is that the user intends the first type declaration and uses the type as though the first declaration were valid, in which case if we assume the second declaration overrides the first we would show many semantic errors incorrectly. Alternatively the user intends the second type declaration to be valid, and if we assume the first is valid we would also show many incorrect semantic errors.
This leaves us with two equivalent scenarios where we could choose to use the opposite type declaration from the user and report many type-related semantic errors incorrectly. We have chosen to use the first declaration of the type when parsing the remainder of the program.

## Calling the ASC interpreter

On a successful parse, we need to call the ASC interpreter. To do this, we have opted to spawn a child process using fork() and then wait for the child to return. The child process, in turn, uses execl() to replace run the ASC interpreter. If the ASC interpreter returns with an exit status, out child process does as well. This way, we can report error conditions in the invocation of the ASC interpreter gracefully.

## What Works

TODO

## Known Bugs

TODO

## Extra Features

TODO

## Implementation Defined Constants

As per ANSI-ISO 7185, the constant maxint is implementation-defined. We have chosen to give maxint the value of 2147483647, as this is the MAX_INT defined in the ASC source code (which makes sense, since it's the upper limit on a signed int in c). We check that integer constants are within the allowable range (-maxint to maxint) when as we parse. For reals, our implementation

handles them as double type internally, and we rely on the error checking of strtod() to make sure the values are valid.

## Generating Labels

From Jonathan's notes, we saw to build a stack of labels for use with IF statements. This stack turned out to be quite handy, as we ended up using it in multiple places. IF statements, WHILE loops, procedure names, and labels for procedure main bodies all use a unique label stack to keep track of things that are (independently) nested.

## Semantic vs Syntax Errors

In a few cases we could choose whether to catch errors at the semantic level or syntax level. Catching them at the syntax level would involve re-writing parts of the grammar (which is why there was a decision to be made in the first place). We usually chose to catch these errors at the semantic level since this involves fewer re-writes of the grammar.
One such decision was whether to catch array invocations involving reals and function calls at the syntax level (for example: 'a = array[1..2e12] of integer') or not. Catching this at the syntax level would involve duplicating large chunks of the grammar (the entire expr tree), which we didn't want to do. Thus, we have opted to catch this type of error as a semantic error.