

## Milestone #2: Front-end

### Overview

The purpose of this milestone is to complete the front-end for the compiler that you will be building throughout the quarter. To complete this milestone you must implement the static semantic analysis phase.

You will be provided with a partial front-end written using `antlr`.

### Static Semantics

After successfully parsing a program, your front-end must validate the program to ensure that it conforms to the static semantics of the language. You should use `antlr` to construct a tree parser that will check the validity of an input program. Your validity checks must include those mentioned in the description of milestone #1.

Your program should report an error if an appropriate `main` function is not defined.

Finally, the check for a proper `return` requires a bit of bookkeeping. To get a feel for what a real compiler might do, your validity check on `return` must recognize each of the following as valid. There are, of course, additional patterns that one could match (such as those that Java does), but we have limited time (though you are free to extend your front-end as you see fit).

```
fun foo(int i) int          # standard case
{
    return i;
}

fun bar(int i) int         # control flow case
{
    if (i > 0)
    {
        return 1;
    }
    else
    {
        if (i < 0)
        {
            return -1;
        }
        else
        {
            return 0;
        }
    }
}
```

Removing any of the `return` statements above should result in an error.

## Command-line Options

Throughout the term, you will extend your compiler to support a number of features. It will be desirable to allow the user to specify if a feature should be used or not. As such, your front-end will support many command-line options. The first such option is `-displayAST`. If this option is present, then your program should display the abstract syntax tree after parsing the input program.