The intermediate representation is done with a series of classes in python which store the AST in member variables. All language structures we have defined in sprint 0 are parsed by the current parser. test.thc contains all language structures we have defined, including nested functions, infix applications, and case statements. datadefs.thc contains a few data declarations, which was written when testing the initial part of the parser that was written. addtwo contains a very simple program which adds two numbers, and is a basic test of the parser.
The parser will print out the parsed program in yaml, as it is a somewhat human readable format. I'd recommend running python3 parser.py sample/addtwo.thc to get an idea of what it does.

You can verify the parser output by reading the yaml and checking that things look correct. Function applications are parsed as repeated applications of a curried function.

addtwo.thc just defines a variable two, which should be evaluated as 1+1.
datadefs.thc defines a Maybe type and a List type.
test.thc defines the above datatypes, and a few functions, some of which use case, then performs a calculation with them in main.

At the moment, the user must specify the types of anything they define. This will make type checking a bit easier for the compiler, although we might implement a more complete type inference later. If we can ensure the types are correct, then the generated code can make many assumptions that will make generating it easier.

I am currently working on what the compiled C code for test.thc should look like, but it probably won't be finished by the deadline. If I do finish it, I will push it in sample/test.c

The relevant python files are lexer.py and parser.py, which are the lexer and parser, as well as thcast.py, which contains the classes that build the AST.

sample/ contains some sample programs for the parser, as well as their expected asts in the corresponding yaml files