# CO221 - Compilers

A compiler is a program which processes programs, including translating a program written in one language (usually higher level) to another programming language (usually in a lower level). In our course, the focus is to generate assembly code from the high level language. This translation goes between high level human concepts, and the data manipulation the machine performs.

Structure

The general structure of a compiler is as follows;

1. **input**   takes in an input program in some language
2. **analysis**   constructs an interal representation of the source structure
3. **synthesis**   walks the representation to generate the output code
4. **output**   creates an output in the target language
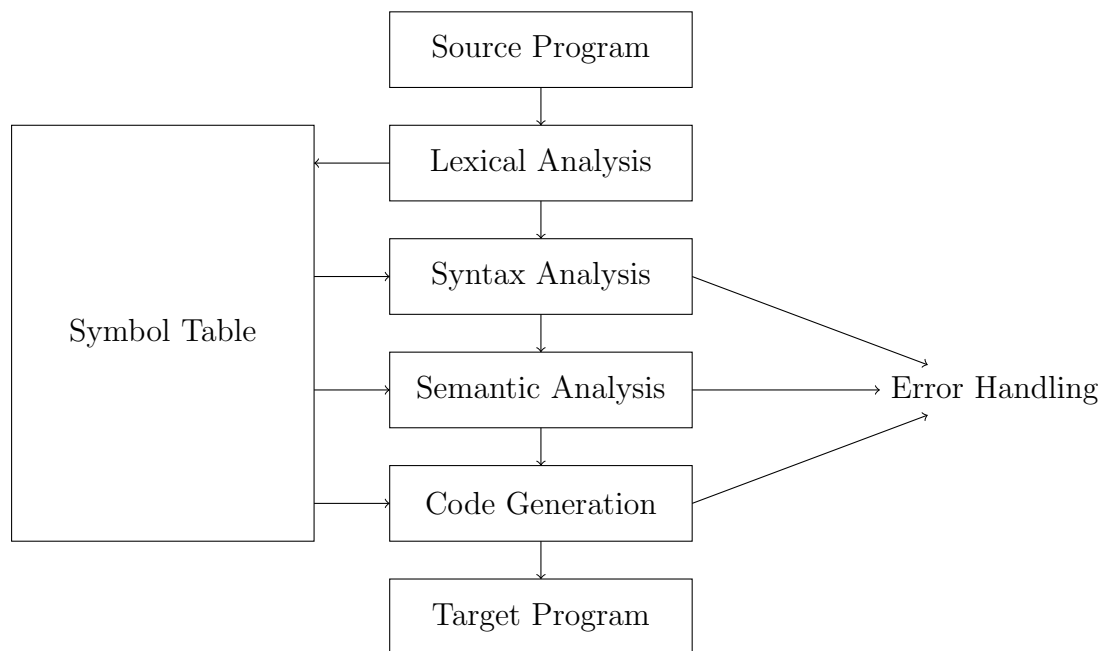
In more detail, it can be represented as follows;
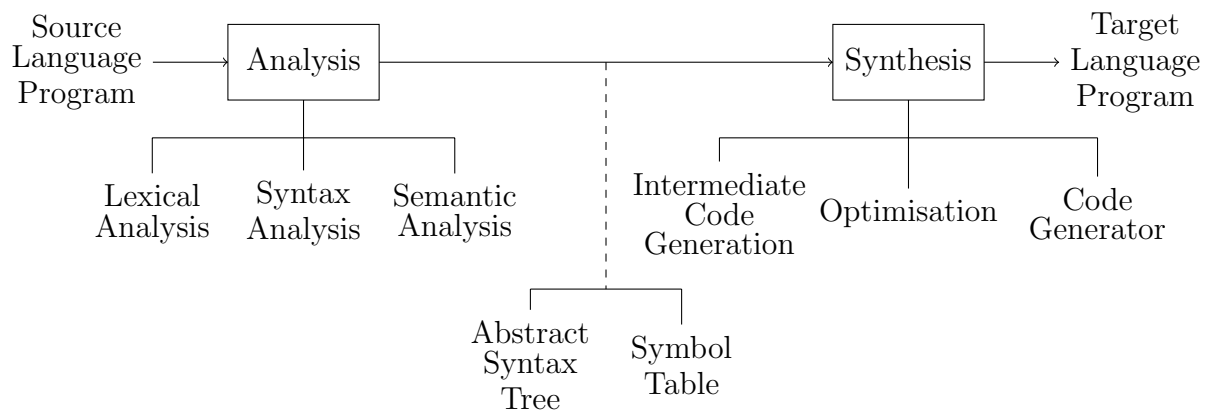
- **lexical analysis** looks at characters of input program, analyses which are keywords (such as converting `if`, and `while` to corresponding tokens), which are user defined words, and which are punctuation, etc.
- **syntax analysis** discovers structure of input
- **semantic analysis** checks that variables are declared before they are used, and that they are used consistently with their types etc.

Simple compilers go straight to code generation, but optimising compilers do several passes of intermediate code generation and optimisation.

The symbol table holds data on variables, such as types. Sometimes we need to know the type of the variable, in order to generate code for the variable, for example if we were to print a variable, it would need to generate different code for strings than it would need to do for integers. Scope rules are also needed.

Phases

Whether all of these phases are done in the order shown is a design choice. For example, lexical analysis and syntax analysis are often interleaved. This can be done when the syntax analysis stage needs the next symbol, and therefore the lexical analysis stage can be used.

Syntax Analysis