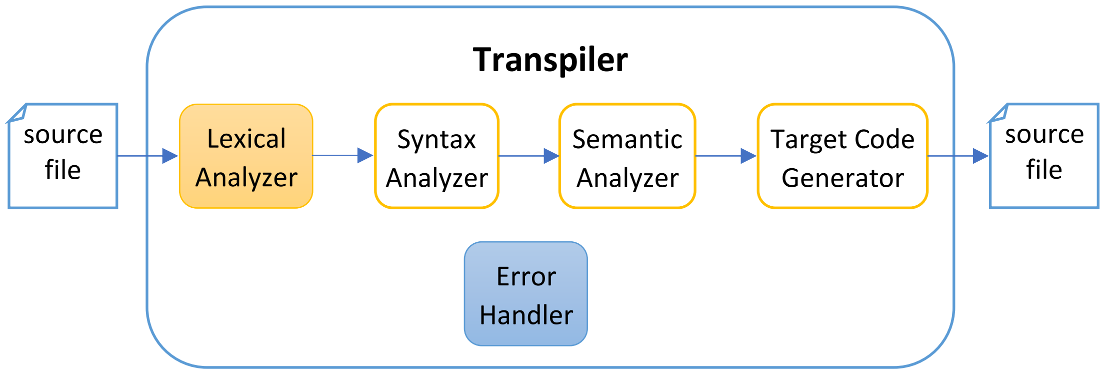
Lab-04 Add a simple lexical analysis

In this lab, we will extend the **transpilers** from lab-03 by adding a simple lexical analyzer—also known as a scanner. In this lab, the compiler contains five components:

1. Lexical analyzer: For this course, we will assume that all words are separated by whitespaces. This assumption allows us to use symbols in the identifier name in the language that we will implement for your term project. For instance, x++ is a legal identifier name for our class but not a legal identifier in C++, Python, etc.
2. Syntax analyzer: For this lab, the syntax will perform the checking on the input to ensure that only the correct input is accepted.
3. Semantic analyzer: For this lab, the semantic analyzer performs the similar tasks as lab-03.
4. Target code generator: For this lab, the target code generator performs the similar tasks as lab-03.
5. Error handler: For this lab, this is an explicit module for taking care of all error reporting.



1 Objectives

1. Construct a compiler incrementally in a back-to-front approach.
2. Extend the transpiler with a simple lexical analyzer and an error handler.

2 Learning Outcomes

By completing this lab, learners should be able to

1. build a simple lexical analyzer.
2. construct a simple compiler with 5 components: lexical analyzer, syntax analyzer, semantic analyzer, target code generator, and error handler.

3 Tasks

In this lab, we will break down the input by the whitespaces into a series of token. A token is a pair of (name, value). For instance, if the input stream contains the following string.

```
9876543210 abc d12 + (
```

Then the output tokens may be (number,” 9876543210”), (identifier, “abc”), (identifier, “d12”), (operator, “+”), and (left-parenthesis). Note that it is up to you to choose whether or not to always store the lexeme in the token. i.e. Either (left-parenthesis) or (left-parenthesis, “(“) is a valid token representation.

In this lab, we will accept a+1, whoareyou? as valid identifier. However, but 1+2 is not a valid word. Because 1+2 is neither a number nor an identifier.

Note that you need to change

1. the syntax analyzer to accepting as input a series of tokens instead of a string. You are free to design what would be the output from the syntax analyzer.
2. No need to spend a lot of time on changing the semantic analyzer and the target code generator because all labs are for you to learn about components of the compiler. We will revisit them later on.