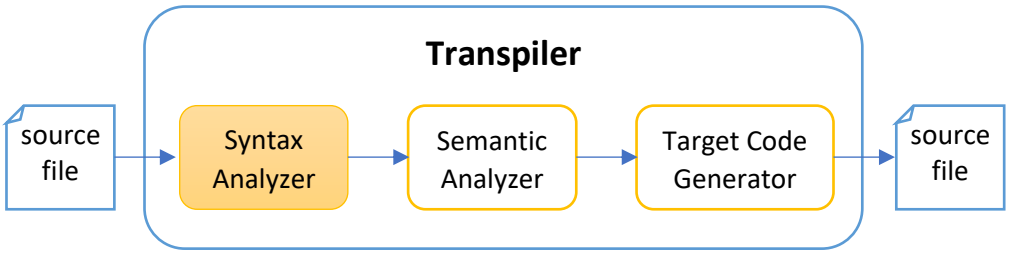
Lab-03 Add a simple syntax analysis

In this lab, we will extend the **transpilers** from lab-02 by adding a simple syntax analyzer. In this lab, the compiler contains only three components:

1. Syntax analyzer: For this lab, the syntax will perform the checking on the input to ensure that only the correct input is accepted.
2. Semantic analyzer: Same as lab-02.
3. Target code generator: Same as lab-02.



1 Objectives

1. Construct a compiler incrementally in a back-to-front approach.
2. Extend the transpiler with a simple syntax analyzer.

2 Learning Outcomes

By completing this lab, learners should be able to

1. build a simple syntax analyzer.
2. construct a simple compiler with 3 components: syntax analyzer, semantic analyzer, and target code generator.

3 Tasks

In this lab, we will check whether the input is an integer. If the input is not a number, display an error and the transpiler should stop. On the other hands, if the input is a number, it should then be passed to the semantic analyzer to check whether it is between 0 – 2,147,483,647, i.e. whether it is a positive 32-bit integer. If the input is not in the specified range [0, 2147483647], the program should inform the user that the number is out-of-range. The transpiler should generate the target codes similar to lab-01. You should test the program to ensure that if the input is a number, it should pass through the syntax analyzer without problems.

Note that it depends on how you implement the transpiler in lab-02.

1. When printing out the error, you may want to also print the name of the phase that the error occurs. For instance,

   [Syntax Analysis], ‘a’ is not a number.
   [Semantic Analysis], 9876543210 is not in range 0 – 2,147,483,647.

2. You may want to implement the error handling as a separate module so that the code about error handling will not be spread out in many modules. In other words, we are using the separation-of-concern concept from the aspect-oriented programming.