

IN2031 Oblig 1: AeroScript Expressions

Deadline: September 15th 2026 at 23:59

1 Objective

The goal of the overall project is for you to implement a domain-specific programming language from a specification. This language, called AeroScript, will focus on expressing flight patterns for drones, allowing us to program — at a high level of abstraction — how a drone should move during a flight mission. In your first assignment, you will be implementing a lexer, a parser, and an evaluator for the core *expressions* of AeroScript.

2 Syntax of Expressions in AeroScript

The syntax for expressions in AeroScript is given in Figure 1. Expressions are either unary negation, binary (infix) operations, random ranges, points, real (in practice, floating point) numbers, or expressions in parenthesis. A range and a point is the compound of two expressions with square brackets and parentheses respectively.

$n \in \mathbb{Q}$	
<i>expression</i>	<code>:= 'neg' expression expression op expression 'random' range?</code>
	<code> 'point' point n '(' expression ')'</code>
<i>range</i>	<code>:= '[' expression, 'expression']'</code>
<i>point</i>	<code>:= '(' expression, 'expression')'</code>
<i>op</i>	<code>:= '+' '-' '*'</code>

Figure 1. The syntax for expressions in AeroScript. The question mark ($\odot$) denotes choice (zero or one occurrence).

3 Semantics of AeroScript Expressions

AeroScript expressions employ the usual, well-established precedences and associativities for binary operators and negation. For example, multiplication ($*$) binds tighter than addition and subtraction, and negation binds tighter than multiplication. Unary operations binds tighter than binary operations.

The expression `random` evaluates to a random number in the (optionally) provided range, but is undefined for scalars (i.e., numbers) and points.

When binary operators or negation are applied to scalars, all is well; applying addition and subtraction to two points is also well-defined. Further, multiplying a point by a scalar simply multiplies each component of the point by the scalar, and negating a point just negates each component of the point. However, any other operation on points is *undefined*.

4 Your task

The assignment includes a template for the implementation, which contains a starting point the abstract syntax tree (AST) for AeroScript expressions, with a few methods that need an implementation. Thus, your task is the following:

1. Complete the grammar according to the description in this document
2. Complete the code for the AST nodes.
3. Implement a lexer and parser for AeroScript expressions using ANTLR.
4. Extend the code to print AeroScript expressions.
5. Extend the code to evaluate AeroScript expressions. Undefined operations should throw runtime errors.

6. Make sure your code can read, parse, build an abstract syntax tree (AST), and evaluate the expressions in the provided `expressions.aero` file.
7. Test your solution using the handed out JUnit suite. Also, make sure that exceptions are thrown correctly for undefined operations. Consider expanding the test suite.
8. Write a report where you discuss the challenges you faced and choices you made during implementation. Be sure to read the report guide on the [course page](#) first.

5 Considerations for your implementation and report

The following are things to consider when designing your solution. Design choices and solutions to this can be discussed in the report.

- How can the implementation be made *robust*; that is avoid unintended runtime errors? How is I/O best handled?¹
- What does *undefined* mean in relation to the semantics? How should your program handle this?
- What is relevant to test? Why have you made a good test?

This is by no means a complete list. There are more design choices, which can be discussed in your report.

6 What to Submit

You should hand in a `zip` file containing your report (as a PDF) along with the project containing your implementation. Make sure to clean (run `./gradlew clean`) any unnecessary garbage files before zipping.

7 Where to Submit

In IN2031, we use Devilry, not Canvas, for assignments. Go to

<https://devilry.ifi.uio.no>

and log in with your user UiO-name and password. When you are logged in, it should be fairly obvious how to upload the report and code.

¹ “If we fail, fail in the most beautiful way”; *Jürgen Klopp*.