COMP3012 COURSEWORK 2025/26

Introduction

This coursework consists of building a non-optimising compiler for a FORTRAN-like language, targeting the Triangle Abstract Machine. You will use ANTLR as a tool to automatically generate a suitable scanner and parser and design your own data structures for converting the results of parsing into acceptable machine code.

The coursework is with 40% of your overall module mark.

Specification

Source language

The source language for your compiler is called Nottran, which is a stripped-down version of Fortran 90 [1]. This language is specified in a separate document on Moodle.

Target platform

The target for your compiler is bytecode for the Triangle Abstract Machine (TAM) described by Watt and Brown [2]. You will be given a working TAM emulator to test your compiler's output [3]. There are no intentional differences between Watt and Brown's specification and the implementation, but if there are, you should tailor your compiler to produce code that works with the emulator, not the specification. This mirrors real world compilation: if you discover a bug in a particular chip, there's no way of getting the manufacturer to update all the chips already shipped to consumers.

By default, you should expect to output a working binary file. You may choose to output TAM assembly instead, but this will involve a penalty.

Technologies

Your compiler will be written in Java 21 or later and will use ANTLR 4 for scanner and parser generation. Your project should be buildable using Gradle, and an example Gradle configuration will be provided.

Features

The complete set of language features you are expected to handle is listed in the coursework form (see Moodle). Essentially, implementing more complex features will lead to higher marks. I **do not expect** anyone to manage to implement the entire Nottran language as specified, the full language is simply described for completeness.

Marking

Your compiler will be marked by running it against a series of Nottran programs and checking that the compilation results are as expected. You will not see the specific programs used, but you will be provided with similar ones for testing.

As a guide, the table below indicates what grade you might expect based on the set of language features you have implemented. I would expect you to complete **all** the features of a given level before moving on to the next level above (i.e. to get a II-1 you must have implemented **all** the features from II-2 and III).

Class	Features
III	Relational operators Read/write integers and logicals If statement If-then-else statement
II-2	*All the above* Integer variables and assignment Logical variables and assignment Arithmetic operators If statement If-then-else statement Do-while statement Exit statement
II-1	*All the above* Character variables and assignment Read/write characters Array variables, assignment, and indexing Do statement
I	*All the above* Concatenation operator Subroutines Functions Pointer variables and allocation/deallocation * Defined types and defined type variables * Optimisations

Features in I class marked with an asterisk are likely especially difficult. Only attempt them if you are very confident in your compiler:

- **Read/write reals:** Your compiler will have to effectively parse user input to verify that a correctly formatted real number has been entered
- **Defined types:** You will need to dynamically modify your type checker at runtime to accommodate these user-defined types and make sure they can be used appropriately.

If you choose to output assembly instead of making an actual binary file, you will lose 5%.

Submission

The coursework deadline is Thursday 4 December 2025 at 3pm. The standard late penalty of -5% per working day late will be applied.

You will submit your compiler on Moodle as a ZIP/TGZ file of your Gradle project, along with a checklist indicating which language features you believe you have implemented.

References

[1] *Fortran 90*, ISO/IEC 1539:1991, 1991. Retrieved from https://wg5-fortran.org/N001-N1100/N692.pdf.

[2] David A. Watt and Deryck F. Brown. 2000. *Language Processors in Java: Compilers and Interpreters*. Prentice Hall, Harlow.

[3] Ian A. Knight. 2025. triangle-abstract-machine. Retrieved from https://github.com/pszik/triangle-abstract-machine.