THE UNIVERSITY OF NEWCASTLE AUSTRALIA

School of Information and Physical Sciences
COMP3290 Compiler Design
Semester 2, 2025
Project Part 3 (Due: October 26th, 2025)

IMPORTANT!

- This assignment is the continuation of the Part 2 and to be completed by the same group completing Part 1 and 2. In other words, the scanner and parser you designed in Part 1 and 2, respectively, will be augmented with the semantic analyser and code generator for finishing your compiler.
- See Canvas for details about the policy and procedure for groupwork management in this course.
- Note the submission instructions which are different for group leader and group member. See the **Submission** section below.
- Like in Part 1 and 2, for completing Part 3, you are encouraged to use Generative AI (GenAI) tools. Please check the document "Use of Generative AI in Assignment 3" for information, explanation and possible use cases on how such tools are to be used in preparing the solution before you start working on this assignment.

These are each of the subsequent phases of your CD25 compiler for the SM25 machine.

Two parts of this assignment Part 3A and Part 3B are submitted together for grading.

Project Part 3A Semantic Analyser for CD25 language

While logically a different phase, the simplest approach is for most of your semantic analysis to be done as you are building the Syntax Tree, and so will likely be done by adding functionality to your parser. This has been discussed in the lecture and workshop.

This phase determines which syntactically correct CD25 programs can be passed on to the code generator for translation into SM25 object code.

Project Part 3B A CD25 Code Generator for the SM25 Stack Machine

Only those CD25 programs that survive previous phases (*Parts 1, 2 and 3A*) with no errors should be passed on to Part 3B for Code Generation.

Please note that the due date is the end of **Week 12** so you should plan your workload with your examination timetable in mind.

1. Part 3A (weight: 40%)

Output:

The only output for Part A is a program listing (*to file* – in the same folder as the compiler) and any errors for CD25 programs with semantic errors (to the terminal). It should be able to report both lexical errors and syntax errors without crashing that specify the *type of error* and *line number* as a minimum.

Additionally, you will output the number of errors found. Suggested formats for these errors are:

```
Syntax Error: line 27 – invalid if-statement (missing Boolean expression)
Semantic Error: line 54 – no return statement for function (int myFunction (int, int))
Found 2 errors
```

If no errors are found, you will simply report:

```
No errors found
```

…and the program should continue to **Code Generation**.

Semantic Checks Required:

The following *Semantic Checking* is to be implemented:

- CD25 program name at the beginning and at the end must match
- `<id>` names (arrays and variables) must be declared before they are used;
- array size – must be known at compile time;
- strong typing exists for real variables, real arrays, Boolean expressions, and arithmetic operations (such as `numeric ^ INTEGER`);
- valid assignment operations;
- actual parameters in a procedure or function call must match the type of their respective formal parameter in the procedure definition;
- the number of actual parameters in a procedure call must be equal to the number of formal parameters in the procedure definition;
- a function must have at least one return statement.

Additionally:

- <id> names must be unique at their particular function level (scoping)

Suggested target completion date is **October 10th** in order to give yourself plenty of time to complete Part 3B.

2. Part 3B (weight: 50%)

Output:

Output for Part B will produce a valid SM25 Module File (using the *same filename as the source*, with the extension **`.mod`**) – assuming a valid source file is provided – in the same directory as the compiler, as well as a listing (with the extension **`.lst`**) to the same location.

For example, the source file `test.cd` will compile to a Module File called `test.mod`

Additionally, the contents of the compilation will be displayed in the terminal window. The output (both to file and on-screen) will be formatted as per section ***The Structure of the Module File*** from the ***SM User Guide*** document.

The contents of the Module File will look similar to the form:

```
6
41 03 52 91 00 00 00 00
61 43 91 00 00 00 08 61
43 91 00 00 00 16 81 00
00 00 00 81 00 00 00 08
11 43 00 00 00 00 16 64
67 72 00 00 00 00 00 00
.
.
.
```

If compilation is successful, you will report a short message to the terminal stating that compilation has been successful; such as:

```
module-name compiled successfully
```

…where **`module-name`** is replaced with the name of your module file (without the **`.mod`** extension). Any compilation errors should also be reported to the Terminal, along with a message that informs the user that the compilation failed.

3. Group Report (weight: 10%)

Accompanying your submission (regardless of whether or not your group attempted only Part 3A, or both Part 3A and Part 3B) will be a Report. Your report will contain the following sections:

a) Any *changes*, *refactoring*, or *refinement* of the language that have been made (fixing *Left Recursion issues*, attempts to bring the grammar closer to *LL(1)*, etc.) through the project.
b) A *Semantic Analysis* Overview detailing:
    - what has been successfully implemented, and the approach used;
    - what Semantic Analysis has been attempted;
    - what Semantic Analysis is not complete or unsuccessful;
    - any assumptions made, with respect to Semantic Analysis.

c) Details of what aspects of Code Generation have been:
    - implemented successfully;
    - implemented partially (please attach sample code to demonstrate);
    - not implemented;
    - any Machine-Independent Optimisations you have implemented (*such as constant folding*).

d) A ½ to 1 page overview on what extensions you feel would be useful to both the Language (CD) and the Machine Environment (SM).

Your report should be *formatted as a report*! This means you will include *name* and *student number* of each group member, appropriate *headings*, and *address the above criteria*.

Your report should be between two and four A4 pages in length.

Marks will also be awarded in part for how closely your assessment of your implementation matches what has been accomplished – *ie: tell the truth, get better marks in both the report, and the project parts.* ***Obviously the inverse also applies***.

4. Report on the use of GenAI

If you have used Generative AI tools in preparing this assignment, you should write a 2 A4 page reflective report on that. In your reflective note, you should provide a detailed account of your experience and should include, but not limited to, the following elements – (i) overview of the tool(s) used (ii) how the tool(s) was/were integrated in your workflow (iii) what was the quality/correctness of the response generated by the tool(s) and/or how the GenAI tool(s) improved the quality of your solution (iv) how the tool(s) improved your learning experience and problem solving skills (v) what challenges you faced in using those tools and how you solved those challenges (vi) how useful was the tool for your productivity e.g. how much time was saved by using GenAI tool(s) (vii) any ethical consideration or concerns in using GenAI tools in this assessment etc. Check the document “Use of Generative AI in Assignment 3” for more details on the format and content of this report.

5. Submission Requirements

Compilation and Execution:

Please ensure that your project can be compiled on the standard *University Lab Java environment* (*this is currently **Java 21.0)***.

Part 3A and Part 3B will be compiled using the command line: **`javac CD.java`**
Part 3A and Part 3B will be executed using the command line: **`java CD source.txt`**
where **`source.txt`** will be the filename of a CD25 source file specified by the end user (note also, it *may or may not be a txt file*); **do not hardcode this filename**.

Testing Your Compiler:

You are responsible for making up sufficient data files which will adequately test your parser. You are encouraged to exchange CD25 source files, in order to thoroughly test your programs.

Like previous parts, it is recommended that you plan out your attack on this project; don't write the whole thing and then go looking for bugs – you will finish up with a mess, impossible to read, understand and extend later. A short while writing (*henceforth useless*) debug routines will probably save you lots of time later.

ReadMe File:

Your submission for parts 3A and 3B should include a ReadMe file that contains those aspects of the Semantic Analysis and Code Generation that you believe that you have implemented, and *specifically it should list those aspects which you know that you have not implemented*. This can simply be extracted from your formal report.

If I do not know something is not working, I cannot effectively test other aspects, and your final mark will suffer – conversely if I know what to work around, I know what to expect and you'll get better marks.

You may include CD25 source files that specifically demonstrate certain aspects of your compiler that are working (both for Part 3A and for Part 3B).

Submission:

Project Part 3A & B, is due on **Friday October 26th at 23:59pm**.

Although the project is to be completed in group, each student should submit his own assignment. Note the difference between the submission of the group leader and group member.

Group Leader: Group leader should submit the group project (Part 3), group reports (Group Report, Group pre-action plan and meeting minutes) and confidential individual reports (Activities report and Peer evaluation) and individual report on the use of GenAI tools (if applicable).

Zip up all your files and submit them via the Assessment 3 (Part 3) Submission Point within the assessments tab on Canvas. Leave your Part 3 source files in the root. Your Submission Zip file will be named with your student number (eg. **`c1234567.zip`**).

Additionally, use (1) the subfolder **`/GroupReports`** in the root containing (i) Group Report (item 3 above) (ii) Group's Pre-action plan (iii) Meeting minutes and (2) the subfolder **`/PersonalReports`** in the root containing (i) individual activity report (ii) peer evaluation and (iii) Report on the use of GenAI (if applicable). Please note that individual activity report and peer evaluations are confidential.

Thus, the Group Leader will submit their personal reports with the main submission in their .zip file.

Group Member: Group member should submit only confidential individual reports (Activities report and Peer evaluation).

Zip up all your files and submit them via the Assessment 3 (Part 3) Submission Point within the assessments tab on Canvas. Use a subfolder **`/PersonalReports`** in the root containing (1) individual activity report, (2) peer evaluation and (3) Report on the use of GenAI (if applicable). Please note that individual activity report and peer evaluations are confidential. Your Submission Zip file will be named with your student number (eg. **c1234567.zip**).