Assignment 3 Description

Assignment 3 - Jack Compiler

Weighting and Due Dates

- Marks for this assignment contribute 20% of the overall course mark.
- **Hurdle Requirement: If your mark for Assignment 3 is less than 20%, your overall mark for the course will be capped at 45 F.**
- Marks for functionality will be awarded automatically by the web submission system.
- **Due dates: Milestone - 11:55pm Friday of week 11**, **Final - 11:55pm Monday of week 13**.
- **Late penalties:** For each part, the maximum mark awarded will be reduced by 25% per day / part day late. If your mark is greater than the maximum, it will be reduced to the maximum.
- **Core Body of Knowledge (CBOK) Areas:** abstraction, design, hardware and software, data and information, and programming.

Project Description

In this assignment you will complete a variation of projects 10 and 11 in the nand2tetris course, reworked descriptions of **Nand2Tetris Projects 10 and 11** are shown below. In particular, you will write the following programs that are used to implement different components of an optimising Jack compiler that compiles a Jack class into Hack Virtual Machine (VM) code:

- **parser** - this parses a Jack program and constructs an abstract syntax tree.
- **codegen** - this takes an abstract syntax tree and outputs equivalent VM code.
- **pretty** - this takes an abstract syntax tree and produces a carefully formatted Jack program.

- **optimiser-r^** - this copies an abstract syntax tree and removes redundant code where possible.
- **optimiser-e*** - this copies an abstract syntax tree and evaluates expressions where possible.

Notes:

- **^**Only for students enrolled in the undergraduate offering, COMP SCI 2000.
- ***Only for students enrolled in the postgraduate offering, COMP SCI 7081.

SVN Repository

Note: this assignment assumes that you have already created directories for every assignment, workshop, project and exam in your svn repository, as described on the **Startup Files for Workshops and Assignments (https://myuni.adelaide.edu.au/courses/64329/pages/startup-files-for-workshops-and-assignments)** page.

1. If required, checkout a working copy of the assignment3 directory from your svn repository.
2. Change directory to the working copy of the assignment3 directory.
3. Copy the latest startup files from the "**View Feedback**" tab of the "**Assignment 3 - Submit Here**" assignment into the updates sub-directory. **Do not unzip the file.**
4. Run the following command to place the assignment's startup files in the correct locations, this will automatically add them to svn and then commit them to your repository:

```
% make install
```

5. Goto the Web Submission System and make a submission to the "**Assignment 3 - Submit Here**" assignment.

Assignment 3 Files and Directories

In addition to the generic **Makefile** and **updates** sub-directory, the assignment3 directory should now contain the following files and directories:

- ***.cpp** C++ source files, you must edit these files to complete the assignment.

- **includes** - this directory contains **.h** files for precompiled classes.
- **lib** - this directory contains precompiled programs and components.
- **originals** - this directory contains the original versions of the ***.cpp** files you are required to edit.
- **tests** - this directory contains a test script and test data.
- **parser** - a script to run your **parser** program.
- **codegen** - a script to run your **codegen** program.
- **pretty** - a script to run your **pretty** program.
- **optimiser-r** - a script to run your **optimiser-r** program.
- **optimiser-e** - a script to run your **optimiser-e** program.

Note: you need to edit the ***.cpp** files to complete this assignment. All the other files are automatically regenerated every time you run **make**, they must not be changed or added to **svn**.

Note: if a newer version of the startup files is made available, it must be placed in the **updates** sub-directory and added to **svn**. The next time **make** is run, all of the files will be updated except for the ***.cpp** files.

Submission and Marking Scheme

Submissions for this assignment must be made to the **web submission system (https://cs.adelaide.edu.au/services/websubmission)** assignment named: **Assignment 3 - Submit Here**. The assessment is based on "**Assessment of Programming Assignments (https://myuni.adelaide.edu.au/courses/64329/pages/assessment-of-programming-assignments)** ".

Notes:

- the marks for the Milestone Tests will be shown by the **Assignment 3 - Milestone** assignment
- the marks for the Final Tests will be shown by the **Assignment 3 - UG - Final** and the **Assignment 3 - PG - Final** assignments
- the participation marks for this assignment will be shown by the **Assignment 3 - One Week Pre Milestone** and the **Assignment 3 - Pre Milestone** assignments

Your programs must be written in C++ and they will be compiled using the **Makefile** and precompiled components in the **lib** directory. They will be tested using Jack language programs that may or may not be syntactically correct. A wide range of tests will be run, including some ***secret*** tests. **Note**: you will get no feedback on the ***secret*** tests, even if you ask! **Note:** there is a single test script so all component programs will be tested regardless of whether your are enrolled in COMP SCI 2000 or COMP SCI 7081.

Assignment 3 - Milestone Submissions: due 11:55pm Friday of week 11

A mark out of 100 for the Milestone Tests will be awarded by the **web submission system (https://cs.adelaide.edu.au/services/websubmission)** and will contribute up to 20% of your marks for this assignment. The marks awarded will be automatically modified by a code review program, as described on the **Assessment of Programming Assignments (https://myuni.adelaide.edu.au/courses/64329/pages/assessment-of-programming-assignments)** page. The marks for the Parser Tests are used as the marks for the milestone submission.

Assignment 3 - Final Submissions: due 11:55pm Monday of week 13

A mark out of 100 for the Final Tests will be awarded by the **web submission system (https://cs.adelaide.edu.au/services/websubmission)** and will contribute up to 80% of your marks for this assignment. The marks awarded will be automatically modified by a code review program, as described on the **Assessment of Programming Assignments (https://myuni.adelaide.edu.au/courses/64329/pages/assessment-of-programming-assignments)** page.

Your Final Assignment Mark will be the geometric mean of two components, the weighted marks, **MF**, for the Milestone Tests, **M**, and Final Tests, **F**, and a mark for your logbook, **L**. It will be limited to 10% more than the weighted marks, ie

MF = M * 0.2 + F * 0.8

FinalMark = MIN(1.1 * MF, SQRT(MF * L))

NOTE - A logbook mark of 0 results in a Final Submission mark of 0.

Logbook Marking

Important: the logbook must have entries for all work in this assignment, including your milestone submissions. See "**Assessment - Logbook Review (https://myuni.adelaide.edu.au/courses/64329/pages/assessment-logbook-review)** " for details of how your logbook will be assessed.

Assignment Weighting and Workload

The weighting of the tests using each program do not reflect the workload required to complete each program. It is important to recognise this and to make sure that you do not spend an excessive amount of time trying to complete everything at the expense of your other studies. You need to make adequate time for revision for the quiz exams and to not take time away from studying other courses.

The key learning outcomes are best achieved by first completing the parser (52%), which has the highest weight, and then the code generator (28%). The optimiser programs (12%) may take considerably more effort but contribute relatively little to the final grade and may be too challenging for some. The pretty printer (8%) is similar to the code generator. This is a summary of the tests weights for each submission and when combined 20 / 80:

Program	Milestone 20%	Final 80%	Combined 100%
parser	100%	40%	52%
codegen	-	35%	28%
optimiser-e/r	-	15%	12%
pretty	-	10%	8%
Total	100%	100%	100%

Assignment 3 - Participation Marks

Any submissions to this assignment that are made at least one week before the due date for Milestone Submissions may be awarded up to 5 participation marks. The participation marks will be the marks awarded for the Final Tests divided by 20. Your logbook review mark will not affect the participation marks. These participation marks will be allocated to week 12 and will be shown by the **Assignment 3 - One Week Pre Milestone** assignment.

Any submissions to this assignment that are made before the due date for Milestone Submissions may be awarded up to 5 participation marks. The participation marks will be the marks awarded for the Final Tests divided by 20. Your logbook review mark will not affect the participation marks. These participation marks will be allocated to week 12 and will be shown by the **Assignment 3 - Pre Milestone** assignment.

Nand2Tetris Projects 10 & 11: Compiler I & II

Background

Modern compilers, like those of Java and C#, are multi-tiered: the compiler's front-end translates from the high-level language to an intermediate VM language; the compiler's back-end translates further from the VM language to the native code of the host platform. In an earlier workshop we started building the back-end tier of the Jack Compiler (we called it the VM Translator); we now turn to the construction of the compiler's front-end. This construction will span two parts: syntax analysis and code generation.

Objective

In this project we build a Syntax Analyser that parses Jack programs according to the Jack grammar, producing an abstract syntax tree that captures the program's structure. We then write separate logic that can apply any number of transformations to our abstract syntax tree. The transformations may include pretty printing the original program, applying

specific optimisations to the abstract syntax tree or generating VM code. This mirrors the approaches used in the workshops.

Resources

The relevant reading for this project is Chapters 10 and 11. However, you should follow the program structure used in earlier workshops rather than the proposed structure in Chapters 10 and 11. **You must write your programs in C++.** The startup files include test files and test scripts you can use to check your work on this assignment. A set of precompiled classes similar to those used in the workshops and the previous assignment are also provided.

Testing and IO

We have a provided a description of the specific requirements for each component program on its own page. However before starting work on any of the component programs you should review the pages on Testing and IO Controls.

Testing

Details of the test data including the how to review the results of each test are described on the **Assignment 3 | testing (https://myuni.adelaide.edu.au/courses/64329/pages/assignment-3-%7C-testing)** page.

IO Controls

Each component program has specific requirements for what it should or should not output when it is working correctly and what to do when an error occurs. Unless specified otherwise, the default error handling process for each component program is to terminate the program with an exit status of 0 and to have not produced any output. Unfortunately, this can make it difficult to trace the execution of your programs and get meaningful error messages from them during development. To allow you to achieve both, a number of output buffering and error reporting functions have been provided and are described on the **Assignment 3 | io controls (https://myuni.adelaide.edu.au/courses/64329/pages/assignment-3-%7C-io-controls)** page.

Component Programs

parser

The **parser** program uses the provided tokeniser to parse a Jack program and construct an equivalent abstract syntax tree. The specific requirements for this component program are described on the **Assignment 3 | parser (https://myuni.adelaide.edu.au/courses/64329/pages/assignment-3-%7C-parser)** page.

codegen

The **codegen** program traverses an abstract syntax tree to generate virtual machine code. The specific requirements for this component program are described on the **Assignment 3 | codegen (https://myuni.adelaide.edu.au/courses/64329/pages/assignment-3-%7C-codegen)** page.

pretty

The **pretty** program traverses an abstract syntax tree and prints a Jack program formatted to a specific coding standard. The specific requirements for this component program are described on the **Assignment 3 | pretty (https://myuni.adelaide.edu.au/courses/64329/pages/assignment-3-%7C-pretty)** page.

optimiser-r^

The **optimiser-r** program traverses an abstract syntax tree and generates a new abstract syntax tree with redundant code removed if possible. The specific requirements for this component program are described on the **Assignment 3 | optimiser_r (https://myuni.adelaide.edu.au/courses/64329/pages/assignment-3-%7C-optimiser-r)** page.

optimiser-e*

The **optimiser-e** program traverses an abstract syntax tree and generates a new abstract syntax tree with all expressions pre-evaluated if possible. The specific requirements for this component program are described on the **Assignment 3 | optimiser_e**

(https://myuni.adelaide.edu.au/courses/64329/pages/assignment-3-%7C-optimiser-e) page.