Assignment 3 Description

Assignment 3 - Jack Compiler

Weighting and Due Dates

- Marks for this assignment contribute 20% of the overall course mark.
- 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.
- **lint^** - this takes an abstract syntax tree and annotates it with warning and / or error messages.

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/54311/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 newest zip file attached below into the updates sub-directory and add it to svn.
4. Run the following command to place the assignment's startup files in the correct locations:

```
% make install
```

5. Add the .cpp startup files and an empty tests directory to your svn repository:

```
% svn add *.cpp
% svn add --depth=empty tests
% svn commit -m "Assignment 3 Startup Files"
```

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, you can add your own tests too.
- **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.
- **lint** - a script to run your **lint** 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**. You can add extra test inputs to the **tests** directory but those are the only additional directories / files that you may add 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/54311/pages/assessment-of-programming-assignments)** ".

Note: the ***Submit Here*** assignment will show a breakdown of your marks by category but it will always show your total mark as capped at **0**.

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:** 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

The marks awarded by the **web submission system** **(https://cs.adelaide.edu.au/services/websubmission)** for the milestone submission contribute up to 20% of your marks for assignment 3. The marks for the Parser Tests are used as the marks for the milestone submission. Your milestone submission mark, after the application of late penalties, will be posted to the myuni gradebook when the assignment marking is complete.

You can view the Parser Tests marks in the **Milestone** *assignment* but submissions must be made using the ***Assignment 3 - Submit Here*** *assignment.*

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

Your final submission mark will be the geometric mean of the weighted marks awarded by the **web submission system** **(https://cs.adelaide.edu.au/services/websubmission/)**, a mark for your logbook and a mark for your code. It will be limited to 20% more than the marks awarded by the **web submission system** **(https://cs.adelaide.edu.au/services/websubmission)**. See "**Assessment - Mark Calculations** **(https://myuni.adelaide.edu.au/courses/54311/pages/assessment-mark-calculations)**" for examples of how the marks are combined. Your final submission mark, after the application of late penalties, will be posted to the myuni gradebook when the assignment marking is complete.

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

Automatic Marking

The final submission marks awarded by the **web submission system** **(https://cs.adelaide.edu.au/services/websubmission)** for each component program will be weighted as follows:

For students enrolled in **COMP SCI 2000 Computer Systems**:

- **Assignment 3 - Final Submissions - UG**
- **parser** - 30%
- **codegen** - 40%
- **pretty** - 10%
- **lint** - 20%

You can view the weighted marks in the **UG Final** *assignment* but submissions must be made using the ***Assignment 3 - Submit Here*** *assignment.*

For students enrolled in **COMP SCI 7081 Computer Systems**:

- **Assignment 3 - Final Submissions - PG**
- **parser** - 30%
- **codegen** - 40%
- **pretty** - 10%
- **optimiser-r** - 20%

You can view the weighted marks in the **PG Final** *assignment* but submissions must be made using the ***Assignment 3 - Submit Here*** *assignment.*

Logbook Marking

Important: the logbook must have entries for all work in this assignment, including your milestone submissions and all of the component programs. All the logbook links in the **web submission system (https://cs.adelaide.edu.au/services/websubmission)** assignments for Assignment 3 point to the same shared logbook. See "**Assessment - Logbook Review (https://myuni.adelaide.edu.au/courses/54311/pages/assessment-logbook-review)** " for details of how your logbook will be assessed.

Code Review Marking

For each of your programming assignments you are expected to submit well written **code**. See "**Assessment - Code Review (https://myuni.adelaide.edu.au/courses/54311/pages/assessment-code-review)** " for details of how your code will be assessed.

Assignment 3 - Participation Marks

Any submissions to assignment 3 that are made more than one week before the due date for Milestone Submissions may be awarded up to 10 participation marks. The participation marks will be the weighted marks awarded for the Final Submissions divided by 10. You can view the participation marks awarded in the ***One Week Pre Milestone*** *assignment* but submissions must be made using the ***Assignment 3 - Submit Here*** *assignment*. The participation marks will be allocated to week 10.

Any submissions to assignment 3 that are made no later than the due date for Milestone Submissions may be awarded up to 10 participation marks. The participation marks will be the weighted marks awarded for the Final Submissions divided by 10. You can view the participation marks awarded in either the ***Pre Milestone*** *assignment* but submissions must be made using the ***Assignment 3 - Submit Here*** *assignment*. The participation marks will be allocated to week 11.

A maximum of 20 participation marks are available.

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++.** You should use the provided command **diffc** to compare your program outputs to the example output files supplied by us. A set of precompiled classes similar to those used in the workshops and the previous assignment are in the zip file attached below. All the test files and test scripts necessary for this project are available in the zip file attached below.

Testing and IO

We have a provided a description of the requirements for each component program on its own page. This includes instructions on how to compile, run and test each component program. However before starting work on any of the component programs you should review the pages on Testing and IO Controls.

Testing

The test data including the convention used to name expected outputs for each test are described on the **Assignment 3 | testing (https://myuni.adelaide.edu.au/courses/54311/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/54311/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/54311/pages/assignment-3-%7C-parser) (https://myuni.adelaide.edu.au/courses/54311/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/54311/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/54311/pages/assignment-3-%7C-pretty)** page.

lint^

The **lint** program traverses an abstract syntax tree and generates a new abstract syntax tree that has been annotated with warning and / or error messages. The specific requirements for this component program are described on the **Assignment 3 | lint (https://myuni.adelaide.edu.au/courses/54311/pages/assignment-3-%7C-lint)** page.

optimiser-r*

The **optimiser-r** program traverses an abstract syntax tree produced 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/54311/pages/assignment-3-%7C-optimiser-r)** page.

Startup Files

The newest of the following zip file(s) must be placed in the updates sub-directory and added to svn. When make is run, the newest zip file in the updates directory is used to update the startup files. Any files you are required to edit will not be updated but, a copy of the latest version of those files will be placed in the sub-directory originals.

- **assignment3-20201101-194606.zip** **(https://myuni.adelaide.edu.au/courses/54311/files/7703393/download?wrap=1)**
- assignment3-20201021-161557.zip
- assignment3-20201005-113205.zip
- assignment3-20200919-143324.zip
- assignment3-20200912-151729.zip
- assignment3-20200814-101854.zip
- assignment3-20200724-193452.zip