Assignment 3 Description

Assignment 3 - Jack Compiler

Weighting and Due Dates

- Marks for this assignment contribute 7.5% 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 Friday of week 12.
- **Late penalties:**The maximum mark awarded will be reduced by 25% per day / part day late. Marks above the cap will be discarded.
- **Core Body of Knowledge (CBOK) Areas:** abstraction, design, hardware and software, data and information, and programming (see here (Links to an external site.)Links to an external site. for a detailed description of the CBOK).

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 functions that are used by simple driver programs to implement different components of an optimising Jack compiler that compiles a Jack class into Hack Virtual Machine (VM) code:

- **jack_parser()** - this parses a Jack program and constructs an abstract syntax tree.
- **jack_codegen()** - this takes an abstract syntax tree and outputs equivalent VM code.
- **jack_pretty()** - this takes an abstract syntax tree and produces a carefully formatted Jack program.
- **jack_optimiser_r()** - this copies an abstract syntax tree and removes redundant code.

SVN Repository

You must create a directory in your svn repository named: **<year>/<semester>/cs/assignment3**. This directory must only contain the following files and directories - the web submission system will check this:

- **Makefile** - this file is used by **make** to compile your submission - **do not modify this file**.
- **.cpp** C++ source files - naming as specified in the component function requirements.
- **.h** C++ include files - naming as specified in the component function requirements.
- **lib** - this directory contains precompiled programs and components - **do not modify this directory**.
- **includes** - this directory contains **.h** files for precompiled classes - **do not modify this directory**.

- **tests** - this directory contains a test script and test data, you can add your own tests too.

Note: if the **lib.a** and **.o** files in the **lib** directory do not get added to your svn repository you will need explicitly added them using:

% svn add lib/lib.a lib/*.o

Submission and Marking Scheme

This assignment has two assignments in the web submission system named: ***Assignment 3 - Milestone Submissions*** and ***Assignment 3 - Final Submissions***.

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

The marks awarded by the web submission system for the milestone submission contribute up to 20% of your marks for assignment 3. Your milestone submission mark, after the application of late penalties, will be posted to the myuni gradebook.

Your programs must be written in C++ and will be tested using Jack language programs that that may or may not be syntactically correct and previously generated abstract syntax trees. Although a wide range of tests may be run, including some ***secret*** tests, marks will only be recorded for those tests that require a working **jack_parser()** function. Your programs will be compiled using the **Makefile** and precompiled components in the **lib** directory. This includes the main functions for each test program that uses one of the component functions that you write. **Note**: you will get no feedback on the ***secret*** tests, even if you ask!

Assignment 3 - Final Submissions: due 11:55pm Friday of week 12

The marks awarded for the final submission contribute up to 80% of your marks for assignment 3.

Your mark for the final submission will be either the marks awarded by the web submission system or, if they exceed either the marks awarded for your logbook or the marks awarded for your coding, your final submission mark will be the geometric mean of these three marks. Your final submission mark, after the application of late penalties, will be posted to the myuni gradebook.

Important: the logbook must have entries for all work in this assignment, including your milestone submissions. See "Assessment of Practical Assignments" for the logbook rubric.

Automatic Marking

The automatic marking will run exactly the same tests used for the milestone submission but it will record marks for **all** of the tests. **Note**: you will get no feedback on the ***secret*** tests, even if you ask!

The marks from the automatic tests for each component function will be weighted as follows:

- **jack_parser()** - 30%
- **jack_codegen()** - 40%

- **jack_pretty()** - 20%
- **jack_optimiser_r()** - 10%

The test scripts are able to test each component function independently but, we strongly suggest that you attempt the component functions in the order above so that you get the most return on your effort. You should gain the most significant learning benefit from completing both the **jack_parser()** and **jack_codegen()** functions.

Code Review Marking

The marks for your coding are based on a manual review of your code that will look at your coding style, your commenting and program structure. We expect that all code that you write will be laid out consistently, it will be easy to read, it will have a clear structure, it will not be a single main function, each function will be prefixed by an appropriate comment and any key logic or data structures will also be appropriately commented.

Note: the way in which code is written is extremely important in the real world. One of the major costs of software development is maintenance. That is, taking existing code that needs bugs fixed or modified in order to add new functionality. If code is poorly structured, does not make good use of abstractions, is badly formatted, lacks comments or the comments are confusing, it makes the task of maintaining the code very unpleasant and more error prone than it should.

To reflect the importance of how code is written, the code review assessment may be used to limit your final submission mark, as described above. See "Assessment of Practical Assignments" for the code review rubric.

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 workshop 8 we started building the back-end tier of the Jack Compiler (we called it the VM Translator); we now turn to construct 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 have a choice, we can morph the logic that generates the abstract syntax tree into logic that generates VM code or we can 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 alternative approaches used in workshop 11.

Resources

The relevant reading for this project is Chapters 10 and 11. However, you should follow the program structure used in workshops 8, 10 and 11 rather than the proposed structure in Chapters 10 and 11. **You must write your programs in C++.** You should use the Linux command **diff** 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.

Component Functions

We have a provided a description of the requirements for each component function on its own page. This includes instructions on how to compiler, run and test each component function.

jack_parser()

The **jack_parser()** function uses the provided tokeniser to parse a Jack program and construct an equivalent abstract syntax tree. The specific requirements for this component function are described on the jack_parser() page.

jack_codegen()

The **jack_codegen()** function traverses an abstract syntax tree to generate virtual machine code. The specific requirements for this component function are described on the jack_codegen() page.

jack_pretty()

The **jack_pretty()** function traverses an abstract syntax tree produced and prints a Jack program formatted to a specific coding standard. The specific requirements for this component function are described on the jack_pretty() page.

jack_optimiser_r()

The **jack_optimiser_r()** function traverses an abstract syntax tree produced and generates a new abstract syntax tree with redundant program elements removed. The specific requirements for this component function are described on the jack_optimiser_r() page.

Testing

The test data including the convention used to name expected outputs for each test as described on the Testing page.

Startup Files

- assignment3.zip
-