**Compilers Optimisation Coursework Report**

*Group Members: Galen Han, Alexander Xu, Jamie Law*

**Overview**

This report will deal with explaining the core algorithms and heuristics of our optimisation methods in ConstantFolder.java.

**Algorithms**

**Initalization**

Building upon the original provided code, we obtain the methods from the ClassGen. We then iterate through each method and call upon our optimise method function.

**Optimise method**

In this section, we perform our optimisation (by calling upon other functions) on each method (method one, method two etc. from each unoptimised java file).

Initially, the code is obtained from the method, following which we form the instruction list. A method generator is initialised with the original method as the baseline (in preparation for creating the optimised method).

We apply peephole optimisations on the instruction list repeatedly, until a single iteration is unable to find any additional optimisations. This is done by using a counter and a while loop to perform arithmetic and comparison optimisations. This counter is incremented by the method return types; arithmetic optimisation and comparison optimisation both return an integer indicating how many optimisations have been made. This counter is reset every time the loop is re-entered. If the counter stays at 0 after an iteration, then the loop is exited as this indicates that no more optimisations can be made.

Following this, we ensure jump handles are all within the current method. Then, we set max stack/max locals using the aforementioned method generator. The method is replaced in the original class with optimised instructions by getting a new method from the method generator and replacing the original method with this new method.

**Optimise arithmetic operations**

This method handles arithmetic operations. A counter is used to indicate how many instructions have been optimised (needed in optimise method). In order to identify an arithmetic operation, **regular expressions** are used, which accepts: *ConstantPushInstruction/CPInstruction/LoadInstruction with an optional ConversionInstruction* ***twice****, ending with an ArithmeticInstruction followed by an optional INVOKEVIRTUAL and (IINC GotoInstruction)*

This regular expression covers all arithmetic operations matchings (as well as recognising iterations, such as for loops by identifying ). The program uses an InstructionFinder to iterate through the instruction list and identify arithmetic operations via this regex.

Once an arithmetic operation has been identified, an InstructionHandle is used to

**Optimise comparisons**