CS5990/7100 – Value Numbering

Due Date: *Monday, February 24, 2022 @ 5 pm*

Project Summary: Your task is to write a program that takes as input an `iloc` program representation generated by the front end of a compiler and performs a local value numbering (LVN). You task is to do the following

1. modify the iloc parser to generate a list of iloc instructions (you choose your own abstract representation of Iloc)
2. build basic blocks
3. perform local value numbering on the basic blocks
4. emit optimized iloc code

You may delete redundant expressions, but you may not delete any other instructions.

Running Iloc: The `jar` file `iloc.jar` on MS Teams is an iloc interpreter. To run an iloc program, use the following command:

```
java -jar iloc.jar [-s] [-d] <file>
```

The `-s` option will report the number of instructions executed and the `-d` option puts the interpreter in a command-line debug mode. The debugger supports the following commands:

- `break [<line>|<label>]` - set breakpoint
- `cont` - continue execution
- `del [all|<label>|<line>]` - delete a breakpoint
- `exit` - exit the debugger
- `help` - list breakpoint commands
- `listb` - list all breakpoints
- `list [<label>|<line>|<null>]` - list Iloc source
- `print %vr<n>` - print the contents of a virtual register in integer format
- `printf %vr<n>` - print the contents of a virtual register in float format
- `printm [%vr<n>|<label>|<addr>]` - print the contents of memory in integer format
- `printmf [%vr<n>|<label>|<addr>]` - print the contents of memory in float format
- `prints <label>` - print contents of memory in string format
- `quit` - exit the debugger
- `step` - execute the next Iloc instruction and break

Your code is required to work correctly on all of the iloc files found in the `input` directory. The source code (from a langauge called NoLife) is there also. The language is Pascal-like.

Report: You are to write a report on your optimizer consisting of a table summarizing the following on the set of benchmarks provided:

- the original number of operations executed
- the running time of your optimizer
- the number of operations executed for the optimized code

The table should have the following format:

Benchmark	Original # Instr.	Opt. Time	Opt. # Instructions
...	...	...	...

What to Turn In: You should turn in your project to MS Teams. You create a script to build your optimizer named `build` so that if I type `build` your code will be compiled and linked (or whatever for the language you choose). I have provided an ANTLR grammar for Iloc (`iloc.g4`) provided in MS Teams. I have also provided a skeleton of my optimizer in Java in

`my-cs6810-ssa-optimizer.zip`.

The Java source files are in

`my-cs6810-ssa-optimizer/antlr/src`.

You may use it if you wish, but it is not required.

The input files are contained in the skeleton optimizer. The input files are in

`my-cs6810-ssa-optimizer/input/*.il`.

No matter the implementation language, create a `bash` script named `lvn` that invokes your optimizer and emits the optimized `iloc` to a file with the same prefix as the input file and the suffix `.lvn.il`. Finally, include a PDF copy of your report in your submission.

The Intermediate Code: The iloc intermediate code is the same as the one provided in the *Engineering a Compiler* book with a few changes. The changes can be found in the documentation in `Iloc.pdf` on MS Teams. You will have to deal with function calls and stores to memory in your optimizer. You may assume there will be no aliasing in the code provided.