CS 5500 – The Structure of a Compiler
Homework Assignment #6

- This assignment is **due by 11:59 p.m. on Monday, Nov. 4, 2019**.
- This assignment will be worth **9%** of your course grade.
- A grading rubric is posted on Canvas so that you can see approximately how many points each functional requirement is worth.
- You are to work on this assignment **by yourself**.
- You are strongly encouraged to **take a look at all of the sample input and output files** posted on the Canvas website **before** you actually submit your assignment for grading.

Project Requirements

For this assignment you are to modify your HW #5 program to make it perform some **code optimization** strategies on 3-address code. Your program will have to: **(1) build a directed acyclic graph from 3-address instructions, (2) perform optimizations on the graph, and (3) then translate the graph back to a list of 3-address code instructions.**

Specifically, your program should implement the following steps on the generated 3-address code:

(1) **Identify the basic blocks** using the "leader" algorithm discussed in class.

(2) **For each basic block**, perform each of the following **local optimizations** (in **exactly** the order listed!) until no further changes can be made to the code:

  (a) **constant folding**

  (b) **algebraic simplification** (only for x + 0, 0 + x, 1 * x, and x * 1, where x is a variable or a constant)[1]

  (c) **common subexpression elimination** (note: x * y = y * x and x + y = y + x, where x and y are any variables or constants)

(3) **Eliminate dead code** instructions, where here dead code is only defined as instructions of the form *<tempVariable> = <constant>.*

(4) **Output the 3-address code program** that results from optimization of the basic blocks. In your output, include a line number for each instruction (even the labels) beginning with an instruction "(0)".

Note that you are **not responsible for doing any error checking** (or error recovery) for this program; that is, you may assume that the **input files do not contain any lexical, syntax, or semantic errors**. You also do **not** need to detect **potential runtime errors** like referencing the value of an uninitialized variable.

[1] **In order for us to test algebraic simplification for the x * 1 case, you are to set the width of an integer to 1 (instead of 4, which you were to use in HW #5) when doing array index calculations, and let your 3-address code translator generate expressions that multiply by 1.**

If your *flex* file was named **hw6.l** and your ***bison*** file was named **hw6.y**, we should be able to compile and execute your program on one of the campus Linux machines using the following commands where *inputFileName* is the name of some input file:

```
flex hw6.l
bison hw6.y
g++ hw6.tab.c –o hw6
hw6  <inputFileName
```

Sample Input and Output

Sample input and output files are posted on Canvas. Note that these files were created on a Windows PC, so you should run *dos2unix* on them before using them on a Linux/Unix machine. With the exception of whitespace and capitalization, the output produced by your program **MUST** be **IDENTICAL** to what our "grader" program produces! Otherwise, the grading script says your program is incorrect and the grader has to grade your program manually to try to give you partial credit, which takes a long time ☹

What To Submit For Grading

You should submit via Canvas a single **zip** file containing your ***flex*** and ***bison*** files **as well as any .h files necessary for your solution**. Note that a *make* file will **NOT** be accepted for this assignment (since that is not what the automated grading script is expecting); your *flex* and *bison* files must ***#include*** your .h files as necessary.

Name your *flex* and *bison* files using your last name followed by your first initial (e.g., Homer Simpson would name his files **simpsonh.l** and **simpsonh.y**). Also name your **zip** file accordingly (e.g., **simpsonh.zip**).

Pity Option

This assignment is simple enough that it can be done by manipulating the entries in the list of 3-address instructions you created for HW #5; you would **not have to create a directed acyclic graph** like we have been doing in the lectures on code optimization. However, if you do that, **your grade on this assignment will be lowered by 25%** (i.e., the **maximum** you can get on this project is **75 points**, not 100 points).

Remember that it is more important to get it working in *some simple way* than to do not have it working in a *complex way*!