Lab 3: A Calculator Compiler Middle End---Basic Block and SSA

Lab 2 is due Wednesday May 1st. Your lab report and source code must be submitted by **12:20PM before the class**. The late policy applies to this lab project.

This lab is an individual project. Get started early! The required format for lab reports can be found on the resource page.

Objective: The objective of this project is to implement a calculator compiler middle end. You are going to practice the basic block generation algorithm.

Software tools: You need three open-source software for this project: bison, flex and gcc. They are readily available on almost all computer platforms.

- For this assignment, you can use any Linux machine that has lex and yacc (or flex and bison) installed.
- You may use a Windows 10 machine provided that you install "windows subsystem for linux", which is basically an Ubuntu Linux system.
- You may use a Mac OS X machine provided that you install Xcode.

Specification of the calculator language:

The language: The calculator compiler accepts a language that is very similar to the grammar you handled in the previous lab. Basically, a program written in the calculator language is a list of statement. The only data type allowed is integer. Each statement is an expression that consists of variables, integer constants and operators "+", "-", "*", "/", "!", "**" and "=", as well as parenthesis. Variables don't need to be declared before use, and variable names are case-insensitive. The value of an undeclared variable in its first use is "0". The operator "=" returns the value that is being assigned.

The language also supports conditional expression "?". Its format is "(cond_expression)?(expression)". The conditional expression is similar to the "?" operator in C. The semantic is that the conditional expression will return the value of "expression" if "cond_expression" is non-zero. Otherwise, the whole expression returns "0". An example is "(x=4)?(x=5)".

An example program might look like the following:

```
A = B+C*1
C = (B=D/2)
E=(D=4)?(D=5)
F=D+E
```

Task 1: Generate Basic Blocks

Implement the basic block finding algorithm in your compiler, and print out all basic blocks in a program. The end of the basic block must be "goto"(s) to other basic blocks. For the above example program, an example output can be:

```
BB1:
        Tmp1 = C*1;
        A = B+Tmp1;
        B = D/2;
        C = B;
        D=4;
        If(D){
                goto BB2;
        } else {
                goto BB3;
        }

BB2:
        D=5;
        E=5;
        goto BB4;

BB3:
        E=0;
        goto BB4;

BB4:
        F=D+E;
```

Task 2: Transform into SSA form

Implement the SSA construction algorithm in your compiler, and print out all basic blocks in SSA form in a program. The variables will be renamed, for example, "A" to "A1", "A2", etc. The phi node, in the format of for example "A3=phi(A1, A2);" should be explicitly inserted to the right location. The end of the basic block must be "goto"(s) to other basic blocks. For the above example program, an example output can be:

```
BB1:
        Tmp1 = C0*1;
        A0 = B0+Tmp1;
        B1 = D0/2;
        C1 = B1;
        D1=4;
        If(D1){
                goto BB2;
        } else {
                goto BB3;
        }

BB2:
        D2=5;
        E0=5;
        goto BB4;

BB3:
        E1=0;
        goto BB4;

BB4:
        E2=phi(E0, E1);
        D3=phi(D1, D2);
        F0=D3+E2;
```

What to Turn In

All project source files, **your test files** and report should be submitted to CANVAS by the deadline.