Project 2: Liveness Analysis

Points: 10 out of 30 (total of the projects)
Due Date: Nov. 20
Objectives:

- review the ideas of liveness analysis
- learn how to implement a basic data-flow analysis pass

Tasks:

- **Step-1**: Read the LLVM documents for more references about the APIs.

- **Step-2**: Implement the basic liveness analysis as an LLVM pass (pass/LivenessAnalysis), with the following specifications:

    - given a C function, your implementation finds the **LiveOut** sets for basic blocks in its CFG;
    - your implementation should be able to **handle back edges**, that is, an iterative algorithm (either basic iterative algorithm or the worklist one) should be used.
    - your implementation only needs to handle a **basic scenario**, where

        - all the variables are *local* variables;
        - all the variables and constants are 32 bits signed integers
        - the operators in assignments only include `+, -, *, / (Signed DIV only)`
        - the IR may include comparing and branching instructions, like `icmp` and `br`, which may also ``use'' variables (and ``define'' variables)
        - the IR may include Load/Store instructions

- **Step-3**: Test your implementation to make sure it works correctly. Test cases will be provided by the TA.

Delivery:

- A source code zip package of the implementation (like the one in subfolder pass/LivenessAnalysis)
- Name format: **CS201-20Fall-Project2-YourStudentNo**.zip

Grading Criteria:

- The correctness of the implementation (TA may test your implementation with more test cases);

Input: A C test program created by clang using *.ll format (e.g. test/phase2/1.ll and 2.ll). Here we show the original source code of 1.ll:

```
void test() {
  int a, b, c, d, e, f;
  e = b + c;
  if (e > 0) {
    e = a;
  } else {
    e = b + c;
  }
  a = e + c;
```

```
}
```

Output: The analysis result printed in the **standard error stream** using the format (*basic-block name*: *variable names).* First, it printed the function name ‘test’, then it lists all the live variables reaching the end of each basic block in every single line. The basic block names are generated by clang and we promise each basic block in each test case will have a unique name. For example, the result of 1.ll:

```
--test--
entry: a b c
if.then: c e
if.else: c e
if.end:
```

Reference:

- The LLVM Compiler Infrastructure
- Writing an LLVM Pass