Project 3: Common Subexpression Elimination

Points: 15 out of 30 (total of the projects)
Due Date: Dec. 4
Objectives:

- review the ideas of available express and CSE
- learn how to implement a basic transform pass

Tasks:

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

- **Step-2**: Implement the basic available expression analysis and common subexpression elimination, with the following specifications:

    - given a C program, your implementation should be able to find all available expressions for each function and remove the redundant expressions.
    - your implementation does NOT need to remove the redundant memory copies (load/store), but only need to remove the redundant computations;
    - 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`
        - 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 package of the implementation (like the one in subfolder pass/CSElimination)
- Name format: **CS201-20Fall-Project3-YourStudentNo**.zip

Grading Criteria:

- The correctness of the implementation (TA may test your implementation with more test cases);
- The number of redundant computations eliminated.

Input: A C test program created by clang using *.ll format (e.g. test/phase3/1.ll and 2.ll). Here we show the original source code:

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

}

Output: The newly transformed program printed in the **standard output stream** to eliminate redundancy of expression calculations. In the attached table, it shows another variable %tmp has been created and used to store common variables. In the end, %tmp is used to eliminate the original computation (b+c).

Reference:

- The LLVM Compiler Infrastructure
- Writing an LLVM Pass

INPUT	OUTPUT

```
define dso_local void @test() #0 {
entry:
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  %c = alloca i32, align 4
  %d = alloca i32, align 4
  %e = alloca i32, align 4
  %f = alloca i32, align 4
  %0 = load i32, i32* %f, align 4
  store i32 %0, i32* %c, align 4
  %1 = load i32, i32* %e, align 4
  %cmp = icmp sgt i32 %1, 0
  br i1 %cmp, label %if.then, label %if.else
if.then:

  %2 = load i32, i32* %a, align 4
  %3 = load i32, i32* %e, align 4
  %sub = sub nsw i32 %2, %3
  store i32 %sub, i32* %b, align 4
  %4 = load i32, i32* %b, align 4
  %5 = load i32, i32* %c, align 4
  %add = add nsw i32 %4, %5
  store i32 %add, i32* %e, align 4
  br label %if.end
if.else:

  %6 = load i32, i32* %b, align 4
  %7 = load i32, i32* %c, align 4
  %add1 = add nsw i32 %6, %7
  store i32 %add1, i32* %e, align 4
  br label %if.end
if.end:

  %8 = load i32, i32* %b, align 4
  %9 = load i32, i32* %c, align 4
  %add2 = add nsw i32 %8, %9
  store i32 %add2, i32* %a, align 4
  ret void
}
```

```
define dso_local void @test() #0 {
entry:
  %tmp = alloca i32, align 4
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  %c = alloca i32, align 4
  %d = alloca i32, align 4
  %e = alloca i32, align 4
  %f = alloca i32, align 4
  %0 = load i32, i32* %f, align 4
  store i32 %0, i32* %c, align 4
  %1 = load i32, i32* %e, align 4
  %cmp = icmp sgt i32 %1, 0
  br i1 %cmp, label %if.then, label %if.else
if.then:

  %2 = load i32, i32* %a, align 4
  %3 = load i32, i32* %e, align 4
  %sub = sub nsw i32 %2, %3
  store i32 %sub, i32* %b, align 4
  %4 = load i32, i32* %b, align 4
  %5 = load i32, i32* %c, align 4
  %add = add nsw i32 %4, %5
  store i32 %add, i32* %tmp, align 4
  %6 = load i32, i32* %tmp, align 4
  store i32 %6, i32* %e, align 4
  br label %if.end
if.else:

  %7 = load i32, i32* %b, align 4
  %8 = load i32, i32* %c, align 4
  %add1 = add nsw i32 %7, %8
  store i32 %add1, i32* %tmp, align 4
  %9 = load i32, i32* %tmp, align 4
  store i32 %9, i32* %e, align 4
  br label %if.end
if.end:

  %10 = load i32, i32* %b, align 4
  %11 = load i32, i32* %c, align 4
  %12 = load i32, i32* %tmp, align 4
  store i32 %12, i32* %a, align 4
  ret void
}
```