Compiler Project
Phase 3

Team Members:
- Anvaya B Narappa (an001)
- Arnav Rahalkar (araha008)

The CS Elimination was implemented using the llvm pass function and the Algorithm provided

```
/* CS Elimination algorithm */

/* index */
i = 0 ;
/* for each block */
for (auto &llvm::BasicBlock &basic_block : F){


  for (auto &llvm::Instruction &inst : basic_block){
    /* if the statement is a operation such as S:A = (B OP C )*/
    if (llvm::isa<llvm::BinaryOperator>(inst)){
      /* if BOP is available at entry of this BB */
      bool avai = available_at_entry(&basic_block, i);
      /* get the operands' name and the operation */
      bool nr = not_redefined_bef_S(&basic_block, i);

      if (avai && nr ){
        std::vector<int> def = defs_that_reach_s(&basic_block, i);

        /* create a temporary var */
        std::string t = "t";

        /* replace each statement D = B OP C */
        replace_each_statement(def, t, &basic_block);

        /* replace the right hand */
        // replace_right_hand_side(t);

      }

    }
    i++;
  }
}
F.print(errs());
return true; // Indicate this is a Transform pass
```

The test of this pass function provided the following results

For 1.cc

compiler_project > CS201-F23-Template > test > phase3 > 1.ll.out

```
; ModuleID = '<stdin>'
source_filename = "1.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: noinline nounwind uwtable
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:                                          ; preds = %entry
  %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* %t, align 4
  %6 = load i32, i32* %t, align 4
  store i32 %6, i32* %e, align 4
  br label %if.end

if.else:                                          ; preds = %entry
  %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:                                           ; preds = %if.else, %if.then
  %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
}

```

For 2.c

compiler_project > CS201-F23-Template > test > phase3 > 2.ll.out

```
; ModuleID = '<stdin>'
source_filename = "2.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: noinline nounwind uwtable
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
  store i32 0, i32* %a, align 4
  store i32 1, i32* %c, align 4
  br label %do.body

do.body:                                          ; preds = %do.cond, %entry
  %0 = load i32, i32* %a, align 4
  %add = add nsw i32 %0, 1
  store i32 %add, i32* %c, align 4
  %1 = load i32, i32* %c, align 4
  %2 = load i32, i32* %b, align 4
  %mul = mul nsw i32 %1, %2
  store i32 %mul, i32* %c, align 4
  %3 = load i32, i32* %b, align 4
  %cmp = icmp sgt i32 %3, 9
  br i1 %cmp, label %if.then, label %if.else

if.then:                                          ; preds = %do.body
  %4 = load i32, i32* %d, align 4
  %5 = load i32, i32* %c, align 4
  %mul1 = mul nsw i32 %4, %5
  store i32 %mul1, i32* %f, align 4
  %6 = load i32, i32* %f, align 4
  %sub = sub nsw i32 %6, 3
  store i32 %sub, i32* %c, align 4
  br label %if.end

if.else:                                          ; preds = %do.body
  %7 = load i32, i32* %e, align 4
  %add2 = add nsw i32 %7, 1
  store i32 %add2, i32* %a, align 4
  %8 = load i32, i32* %d, align 4
  %div = sdiv i32 %8, 2
  store i32 %div, i32* %e, align 4
  br label %if.end
```

Conclusion:

The results were incorrect. However, this was good opportunity to learn how to approach Redundancy elimination. The CSE is a complex algorithm and requires evaluating all the available expressions and reaching definitions at every point of the basic block.