1 Exercises

Exercise 1. Consider the following *j--* program `SpimSum.java`:

```java
import spim.SPIM;

public class SpimSum {
    public static int compute(int n) {
        int sum = 0, i = n;
        while (i > 0) {
            sum += i--;
        }
        return sum;
    }

    public static void main(String[] args) {
        int result = SpimSum.compute(100);
        SPIM.printInt(result);
        SPIM.printChar('\n');
    }
}
```

The JVM bytecode for the `SpimSum.compute()` method are listed below, with linebreaks denoting basic blocks.

```
  public static int compute(int);
    Code:
      stack=2, locals=3, args_size=1
         0: iconst_0
         1: istore_1
         2: iload_0
         3: istore_2

         4: iload_2
         5: iconst_0
         6: if_icmple     19

         9: iload_1
        10: iload_2
        11: iinc          2, -1
        14: iadd
        15: istore_1
        16: goto          4

        19: iload_1
        20: ireturn
```

The HIR instructions for the method are listed below.

```
  B0 succ: B1
  Locals: I0
  I0: LDLOC 0

  B1 [0, 3] dom: B0 pred: B0  succ: B2
  Locals: I0 I3 I0
  I3: 0

  B2 [LH] [4, 6] dom: B1 pred: B1 B3  succ: B3 B4
  Locals: I0 I5 I6
  I5: [ I3 I11 ]
  I6: [ I0 I10 ]
  I7: 0
  8: if I6 <= I7 then B4 else B3

  B3 [LT] [9, 16] dom: B2 pred: B2  succ: B2
  Locals: I0 I11 I10
  I9: -1
  I10: I6 + I9
  I11: I5 + I6
  12: goto B2

  B4 [19, 20] dom: B2 pred: B2
  Locals: I0 I5 I6
  I13: ireturn I5
```

a. Draw the HIR flow graph for `SpimSum.compute()`.

b. Suppse that the HIR to LIR translation procedure assigns virtual registers `V32`, `V33`, `V34`, `V37`, and `38` to HIR instructions `I3` `I5`, `I6`, `I10`, and `I11` respectively. How are the Phi functions `I5` and `I6` in block `B2` resolved?

Exercise 2. What optimization techniques would you use to improve each of the following code snippets, and how?

a.

```java
static int f() {
    return 42;
}

static int g(int x) {
    return f() * x;
}
```

b.

```java
static int f() {
    int x = 28;
    int y = 42
    int z = x + y * 10;
    return z;
}
```

c.

```java
static int f(int x) {
    int y = x * x * x;
    return x * x * x;
}
```

d.

```java
static int f(int[][] a) {
    int sum = 0;
    int i = 0;
    while (i <= a.length - 1) {
        int j = 0;
        while (j < a[0].length - 1) {
            sum += a[i][j];
            j = j + 1;
        }
        i = i + 1;
    }
    return sum;
}
```

e.

```java
static int f(int[][] a, int[][] b, int[][] c) {
    ...
    c[i][j] = a[i][j] + b[i][j];
    ...
}
```

where `a`, `b`, and `c` have the same dimensions.

f.

```java
static int f(SomeObject o) {
    return o.x * o.y * o.z;
}
```

where `x`, `y`, and `z` are integer fields in `o`.

2 Solutions to Exercises

Solution 1.

a.

b. The Phi functions `I5` and `I6` in block `B2` are resolved by adding `move` instructions at the end of `B2`'s predecessors `B1` and `B3`, as follows:

```
B1:
  move V32 V33
  move $a0 V34

B3:
  move V37 V33
  move V38 V34
```

Solution 2. a. Inlining

```java
static int g(int x) {
    return 42 * x;
}
```

b. Constant folding and constant propagation

```java
static int f() {
    return 448;
}
```

c. Common subexpression elimination

```java
static int f(int x) {
    int y = x * x * x;
    return y;
}
```

d. Lifting loop invariant code

```java
static int f(int[][] a) {
    int sum = 0;
    int i = 0;
    while (i <= a.length - 1) {
        int j = 0;
        int[] a_ = a[i];
        while (j < a_.length - 1) {
            sum += a_[j];
            j = j + 1;
        }
        i = i + 1;
    }
    return sum;
}
```

e. Array bounds check elimination. Since `a`, `b`, and `c` have the same dimensions, perform the array bounds check, ie, check if the indices `i` and `j` are within bounds, just once instead of three times.

f. Null check elimination. Perform null check on the object `o` just once instead of three times.