Name:

Instructions:

1. Write your name at the top of this page.
2. There are 6 problems in this exam and you have 120 minutes to answer them.
3. This is a closed book exam. The algorithms relevant to the exam problems are provided on pages 3 and 4.
4. To receive full credit, your solution must not only be correct but also show all the steps.
5. Discussing the exam contents with anyone who has not taken the exam is a violation of the academic honesty code.

Problem 1. (10 points) Consider the following *j*-- program:

```
public class Mystery {
    private int f = 42;
    private long g = 1729;

    public int f(int t, long u, int v, long w) {
        int x = t + v;
        {
            Mystery m = new Mystery();
            long y = m.g(u, w);
            int z = (int) y * x;
        }
        {
            Mystery m = new Mystery();
            int y = v * x;
            long z = m.g(w, (long) y);
        }
        return x;
    }

    public static long g(long u, long v) {
        long w = (long) f + g;
        int x = (int) u * (int) v;
        long y = u + w + (long) x;
        return y;
    }
}
```

a. (5 points) Show in detail the compilation unit context for `Mystery`, as built by `preAnalyze()`.

b. (5 points) Show in detail the method/local contexts for `f()` and `g()`.

Problem 2. (10 points) Classify the following casts as identity, narrowing, widening, boxing, or unboxing. What code must be generated to support the cast at runtime? If no code is need, say "No runtime code needed".

a. (2 points)

```
Character v = (Character) '42';
```

b. (2 points)

```
double w = (double) (new Double(Math.PI));
```

c. (2 points)

```
ArrayList x = (ArrayList) (new Object());
```

d. (2 points)

```
Object y = (Object) (new ArrayList());
```

e. (2 points)

```
char z = (char) 42;
```

Problem 3. (20 points) Suppose `a` is an array of integers and `y`, `i` and `z` are integers. For each of the *j*-- statements below, list the instructions generated and show how the runtime stack evolves as those instructions are executed.

a. (10 points)

```
a[i] += y;
```

b. (10 points)

```
z = ++a[i];
```

Problem 4. (20 points) Consider the following intermediate JVM instructions for an *iota* method `int mystery(int x)`:

```
>>> mystery(I)I

  0: ldc 0
  2: istore 1
  4: iload 0
  6: ldc 0
  8: if_icmple 28
 11: iload 1
 13: iload 0
 15: iadd
 16: istore 1
 18: iload 0
 20: ldc 1
 22: isub
 23: istore 0
 25: goto 4
 28: iload 1
 30: ireturn
```

a. (10 points) Show the variable and runtime stack trace for the call `mystery(12)`.

b. (5 points) What value does the above call return?

c. (5 points) What does `mystery(x)` compute and return in general?

Problem 5. (20 points) The LIR instructions for the method `int mystery(int x)` from the previous problem are listed below.

```
>>> mystery(I)I

  B0 (pred: [], succ: [B1]):
  0: load v16 r14 -3

  B1 (pred: [B0], succ: [B2]):
  5: set v17 0
  10: copy v18 v16
  15: copy v19 v17

  B2 (pred: [B1, B3], succ: [B3, B4], LH):
  20: set v20 0
  25: jle v18 v20 B4 B3

  B3 (pred: [B2], succ: [B2], LT):
  30: add v21 v19 v18
  35: set v22 1
  40: sub v23 v18 v22
  45: copy v18 v23
  50: copy v19 v21
  55: jump B2

  B4 (pred: [B2], succ: []):
  60: copy r13 v19
  65: return
```

a. (10 points) Compute the liveUse and liveDef sets for each basic block in the method.

b. (10 points) Compute the liveIn and liveOut sets for each basic block in the method.

Problem 6. (20 points) The liveness intervals for the virtual registers (all except `v18`, `v19`, and `v20`) in the LIR instructions from the previous problem are listed below.

```
  v16: [W 0, 10 R]
  v17: [W 5, 15 R]
  v18: ???
  v19: ???
  v20: ???
  v21: [W 30, 50 R]
  v22: [W 35, 40 R]
  v23: [W 40, 45 R]
```

a. (10 points) Compute the liveness intervals for `v18`, `v19`, and `v20`.

b. (5 points) List the neighbors of the 8 vertices (`v16` — `v23`) in the interference graph G for the method, in ascending order.

c. (5 points) Color the graph G using 3 physical registers and draw the graph using symbols ▲, ★, and ■ for vertices, where the symbols denote the three registers.

Instruction for Boxing in *j--*

```
invokestatic java/lang/T.valueOf:
    (t)Ljava/lang/T;
```

where `t` is the primitive type (eg, `I`) and `T` is the corresponding wrapper type (eg, `Integer`).

Instruction for Unboxing in *j--*

```
invokevirtual java/lang/T.tValue:()t
```

where `T` is the wrapper type (eg, `Integer`) and `t` is the corresponding primitive type (eg, `I`).

Instructions for Assignment Operations in *j--*

	x	a[i]	o.f	C.sf
lhs = y	iload y' [dup] istore x'	aload a' iload i' iload y' [dup_x2] iastore	aload o' iload y [dup_x1] putfield f	iload y' [dup] putstatic sf
lhs += y	iload x' iload y' iadd [dup] istore x'	aload a' iload i' dup2 iaload iload y' iadd [dup_x2] iastore	aload o' dup getfield f iload y' iadd [dup_x1] putfield f	getstatic sf iload y' iadd [dup] putstatic sf
++lhs	iinc x',1 [iload x']	aload a' iload i' dup2 iaload iconst_1 iadd [dup_x2] iastore	aload o' dup getfield f iconst_1 iadd [dup_x1] putfield f	getstatic sf iconst_1 iadd [dup] putstatic sf
lhs--	[iload x'] iinc x',-1	aload a' iload i' dup2 iaload [dup_x2] iconst_1 isub iastore	aload o' dup getfield f [dup_x1] iconst_1 isub putfield f	getstatic sf [dup] iconst_1 isub putstatic sf

Local Liveness Sets

Input: The control-flow graph g for a method
Output: Two sets for each basic block: liveUse and liveDef

```
for block b in g.blocks do
    Set b.liveUse ← {}
    Set b.liveDef ← {}
    for instruction i in b.instructions do
        for virtual register v in i.readOperands do
            if v ∉ b.liveDef then
                b.liveUse.add(v)
            end if
        end for
        for virtual register v in i.writeOperands do
            b.liveDef.add(v)
        end for
    end for
end for
```

Global Liveness Sets

Input: The control-flow graph g for a method, and the local liveness sets
Output: Two sets for each basic block: liveIn and liveOut

```
for block b in g.blocks do
    b.liveIn ← {}
    b.liveOut ← {}
end for
repeat
    for block b in g.blocks in reverse order do
        for block s in b.successors do
            b.liveOut ← b.liveOut ∪ s.liveIn
        end for
        b.liveIn ← (b.liveOut − b.liveDef) ∪ b.liveUse
    end for
until no liveOut has changed
```

Liveness Intervals

Input: The control-flow graph g for a method with LIR, and the global liveness sets
Output: A liveness interval for each register, with ranges and use positions

```
for block b in g.blocks in reverse order do
    int bStart ← b.firstInstruction.id
    int bEnd ← b.lastInstruction.id
    for register r in b.liveOut do
        intervals[r].addRange(bStart, bEnd)
    end for
    for instruction i in b.instructions in reverse order do
        for virtual register r in i.writeOperands do
            intervals[r].firstRangeFrom(i.id)
            intervals[r].addUsePosition(i.id, Write)
        end for
        for virtual register r in i.readOperands do
            intervals[r].addRange(bStart, i.id)
            intervals[r].addUsePosition(i.id, Read)
        end for
    end for
end for
```

Graph Coloring

Input: The control-flow graph g for a method
Output: The same g but with the virtual registers in LIR instructions replaced by physical registers

```
success ← false
repeat
    buildLivenessIntervals()
    buildInterferenceGraph()
    success ← assignRegisters()
    if not success then
        generateSpillCode()
    end if
until success
```

Solution 1.

a.

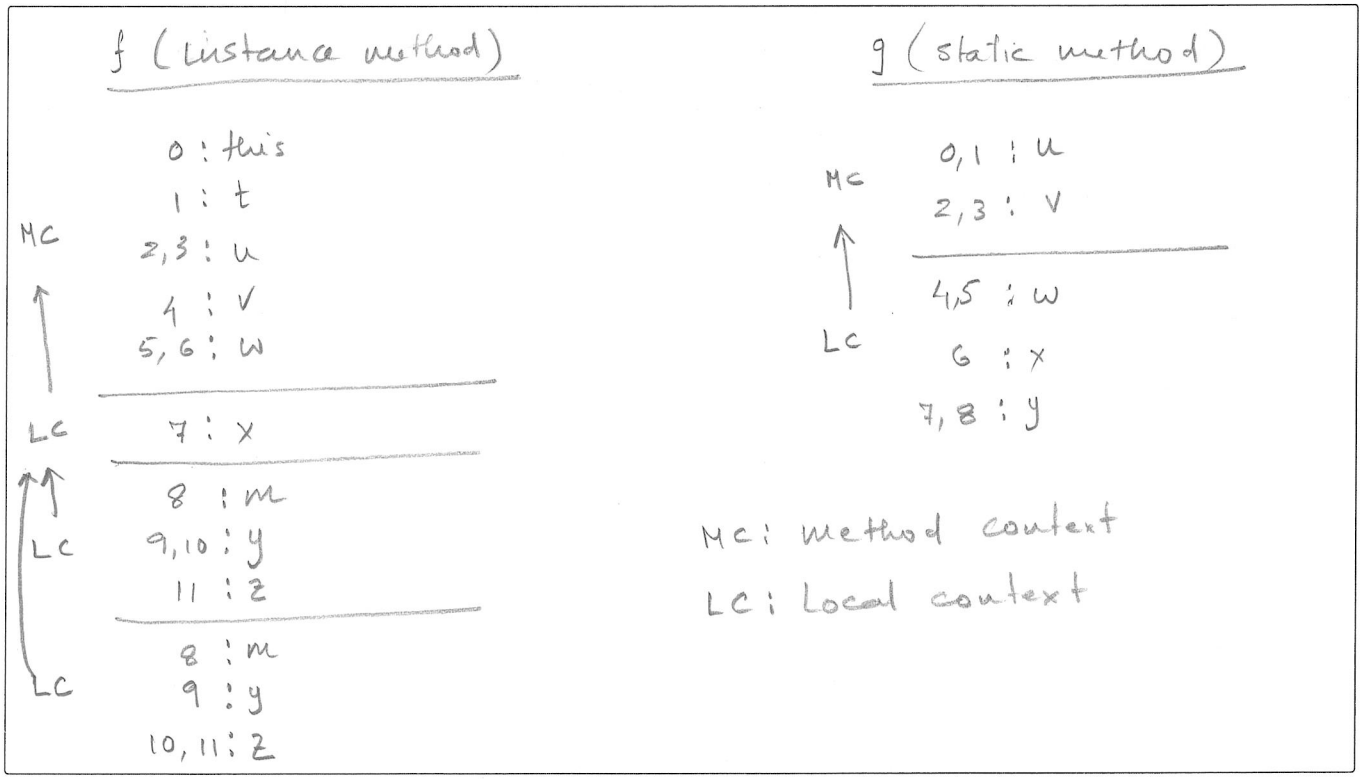
Object
java.lang.Object
→ Object.class

String
java.lang.String
→ String.class

Mystery → Mystery.class (partial)

- private int f
- private long g
- public Mystery()
- public int f(int, long, int, long)
- public static long g(long, long)

b.



Solution 2.

a.

Boxing INVOKESTATIC Character.valueOf: (C)Ljava/lang/Character;

b.

Unboxing INVOKEVIRTUAL Double.doubleValue:(Ljava/lang/Double)D

c.

Narrowing reference CHECKCAST java/util/ArrayList

d.

Widening reference No runtime code needed.

e.

Narrowing primitive I2C

Solution 3.

a.

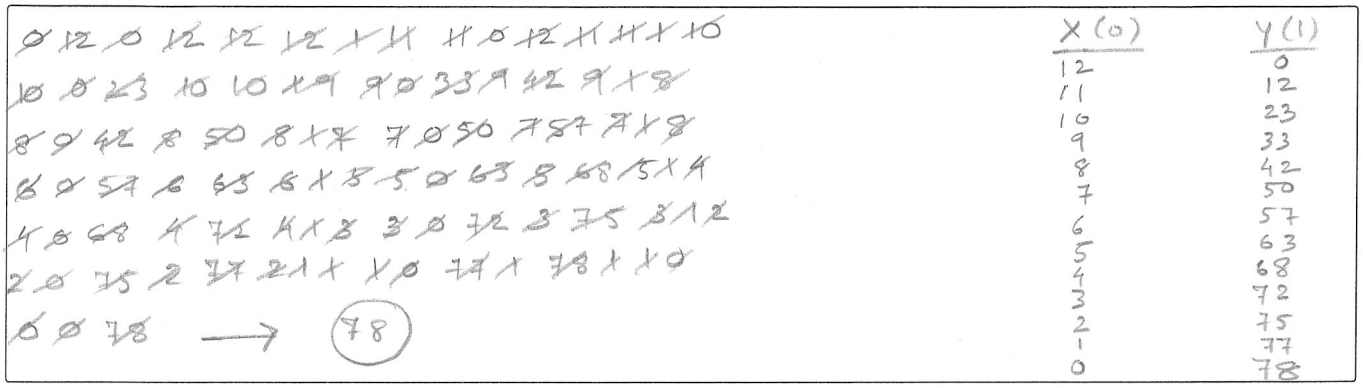
a[i] += y;	aload a	|a
	iload i	|a|i
	dup2	|a|i|a|i
	iaload	|a|i|a[i]
	iload y	|a|i|a[i]|y
	iadd	|a|i|a[i]+y
	iastore	|

b.

z = ++a[i];	aload a	|a
	iload i	|a|i
	dup2	|a|i|a|i
	iaload	|a|i|a[i]
	iconst_1	|a|i|a[i]|1
	iadd	|a|i|a[i]+1
	dup_x2	|a[i]+1|a|i|a[i]+1
	iastore	|a[i]+1

Solution 4.

a.



X(0)	Y(1)
12	0
11	12
10	23
9	33
8	42
7	50
6	57
5	63
4	68
3	72
2	75
1	77
0	78

b.

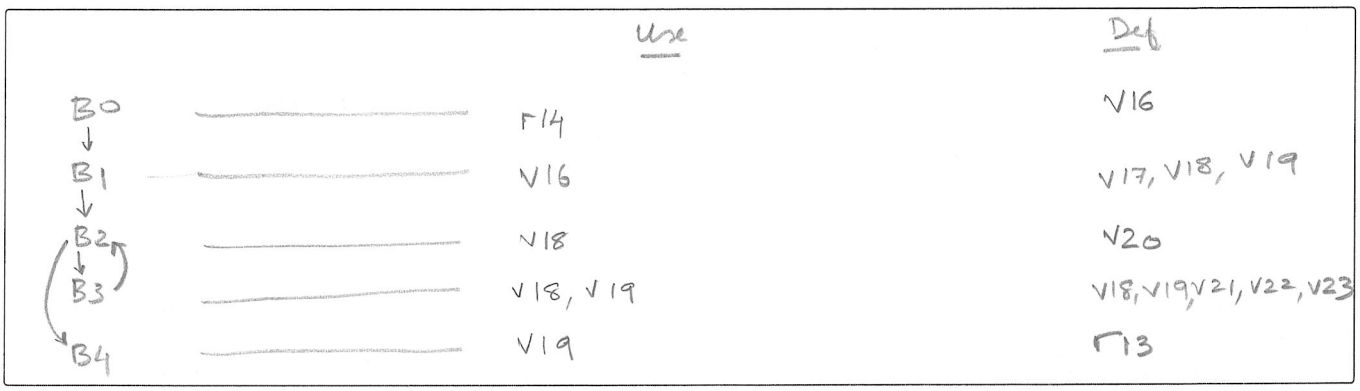
78

c.

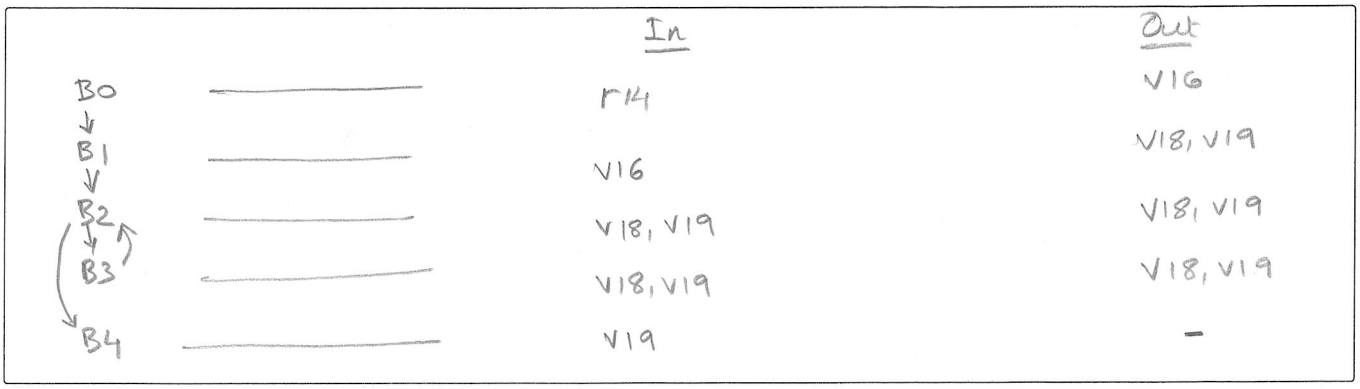
$x + (x-1) + (x-2) + \cdots + 3 + 2 + 1$ (Sum of integers $\leq x$)

Solution 5.

a.

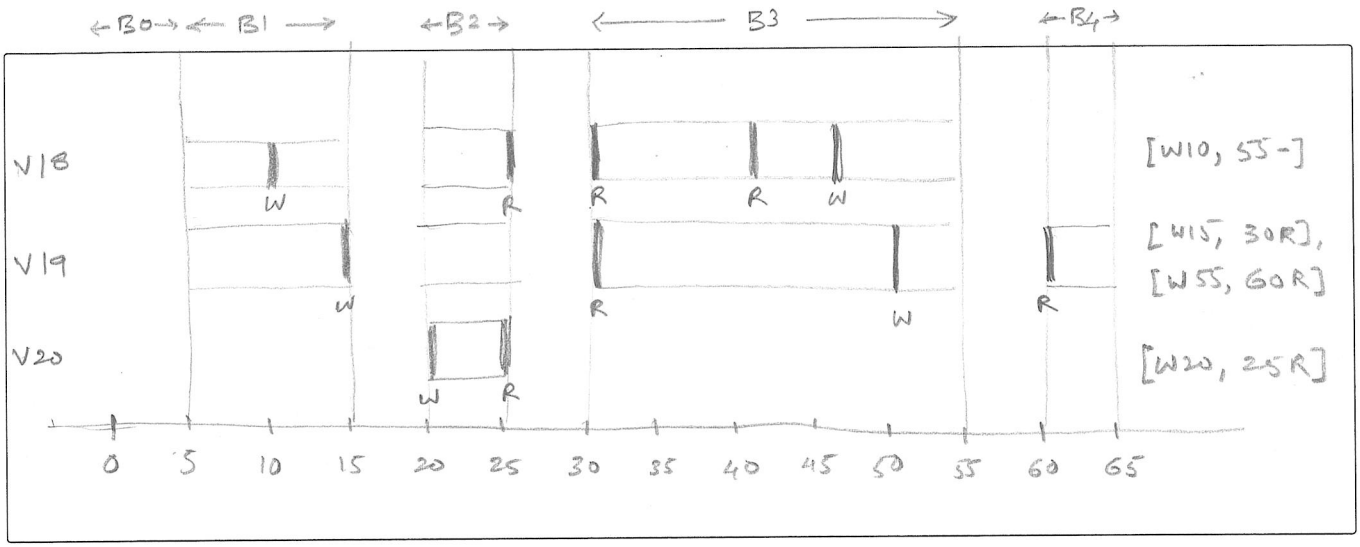


b.

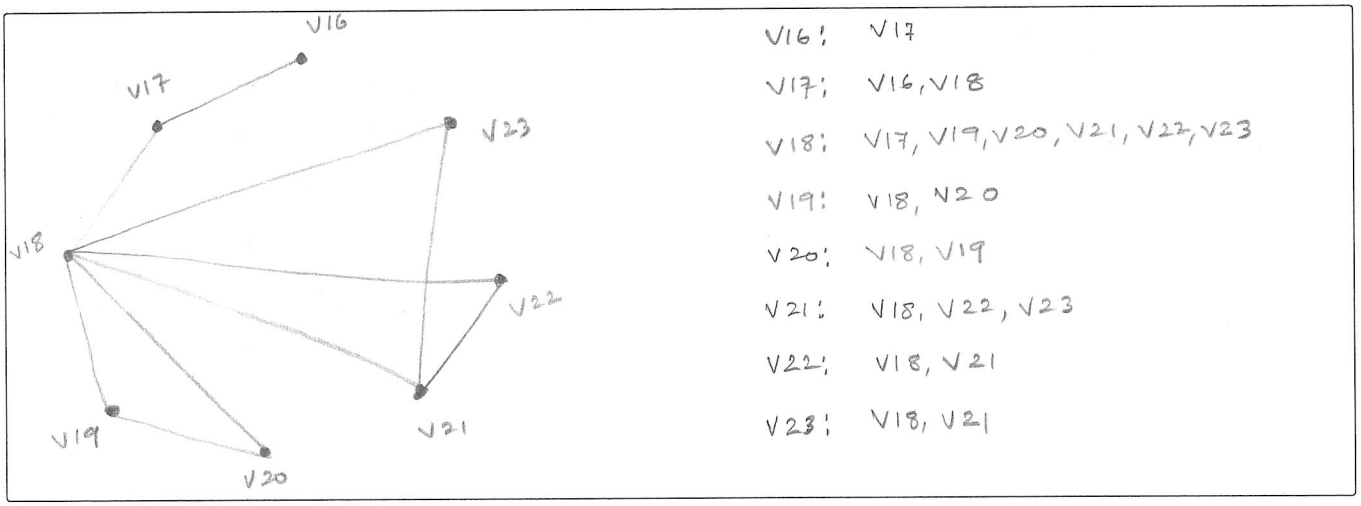


Solution 6.

a.



b.



c.