```
l1: if y = 0 goto l2
    q := x / y
    t := q * y
    r := x - t
    x := y
    y := r
    goto l1
l2: return x
```

Exercise 1: Control flow

(a) Construct the control flow graph for the intermediate code above.

Solution: By line number:

```
1 -> 2 -> 3 -> 4 -> 6 -> 7 -> 1
1 -> 8
```

(b) Show the control flow graph with basic blocks.

Solution:

```
1 -> [2,3,4,5,6,7] -> 1
1 -> 8
```

Exercise 2: Live variable analysis

(a) When is a variable considered live?

Solution: A variable is live if it holds a value that may be observed (read) in the future of the program. The writer of the liveness analysis defines which variables are live at the end of the control flow of a program.

Alternative wording: A variable is live in a program between the point where it was last assigned to and the point where it is read. A variable is also live between the start of the program and the first read of the variable is there is no assignment of that variable in between. The writer of the liveness analysis defines which variables are live at the end of the control flow of a program.

(b) What is the information that live variables analysis calculates and associates with each control flow graph node?

Solution: A set of names of variables that are live at that point in the program.

(c) Is live variable analysis a *may* or *must* analysis?

Solution: It is a *may* analysis.

```
int x = 0;
if(y > 0) {
  x = 1;
```

```
} else {
  y = 0;
}
return x;
```

(d) Given the above MiniJava method body, provide the live variables analysis result per line of code. Explain where the analysis result demonstrates the analysis property (may or must) you answered above.

Solution: In this program variable `x` is live after initialisation despite being immediately reassigned in one branch of the `if` statement. There is a path where `x == 0` is observed, namely when `y > 0`.

```
            // (in , out)
int x = 0;  // {y}, {x,y}
if(y > 0) { // {x,y} , {x} \/ {}
  x = 1;    // {} , {x}
} else {    //
  y = 0;    // {x} , {x}
}           //
return x;   // {x} , {x}
```

(e) What information does live variables expose at assignments, and how can this be used for optimisation?

Solution: At an assignment when the variable that's assigned isn't live, that assignment is not observed by the rest of the program, and is therefore dead code. This dead code can be eliminated in an optimisation step.