Optimising Compilers

(a) Summarise the idea of a basic block and explain why it is useful in intermediate representations for optimising compilers. [3 marks]

(b) Construct the flowgraph (in which every node is a basic block consisting of one or more 3-address instructions) for the C function:

```c
int f(int x, int y)
{
    int r = x + 1;
    if (y == 0) {
        r = r * r;
    } else {
        y = y - 1;
        r = r * y;
    }
    return r + 1;
}
```

[4 marks]

(c) Define static single assignment (SSA) form, and explain the changes you would have to make to your flowgraph from part (b) in order for it to be in SSA form. [3 marks]

(d) Consider a flowgraph in which every node contains a single 3-address instruction. Each node whose instruction assigns some value to a variable is considered a “definition” of that variable; we are interested in discovering, for each node \( n \) in the flowgraph, which definitions reach \( n \). A definition \( m \) is considered to reach \( n \) if the variable to which \( m \) assigns a value may still have that value at entry to \( n \).

(i) Define the notion of a definition reaching a node in the flowgraph in terms of possible execution flows of control. [2 marks]

(ii) By analogy with live variable or available expression analysis, or otherwise, design dataflow equations for computing \( RD(n) \), the set of definitions which can reach a node \( n \). [4 marks]

(iii) Sketch an algorithm to compute \( RD(n) \), briefly commenting on any initialisation required. [4 marks]