

## 12 Optimising Compilers (tmj32)

The following excerpt from a program in C-style code is optimised with a compiler using code-motion transformations. The function `read()` returns a signed integer from the user.

```

10:  a = read();
11:  b = read();
12:  p = &a;
13:  q = &b;
14:  r = &p;
15:  if (read() > 0) {
16:    a = b + 5;
17:  } else {
18:    i = 0;
19:    while (i < 10) {
l10:      c = b + 5;
l11:      **r += *q;
l12:      i += 1;
l13:    }
l14:    a += c;
l15:  }
l16:  print(a);

```

- (a) Describe loop-invariant code motion (LICM) and which expression(s) in the loop above it should move. [2 marks]
- (b) Describe a simple data-flow analysis and a way of using it to identify loop-invariant expressions. Use this to analyse the code above. [5 marks]
- (c) Explain whether all expressions described in Part (a) are found through the analysis in Part (b). [2 marks]
- (d) Describe an analysis that can aid in making LICM more precise in this example. [3 marks]
- (e) Apply the analysis from Part (d) to the code above and redo the analysis from Part (b) to show which expressions described in Part (a) are now found. [4 marks]
- (f) Describe another *code motion* transformation that could be applied to the code after LICM and show the final code after its application. [4 marks]