1. Write out the symbol table(s) (including any attributes that you think are important) for the following piece of code:

```c
int x;
float y;
string z = "hello";

void foo() {
    int x;
    float z;
}

void main() {
    int q;
    string p = "goodbye";
}
```

2. Explain the difference between an L-value and an R-value

3. ECE 468 student Cam Piler has come up with a code generation strategy for his project: while walking his Abstract Syntax Tree, whenever he sees a variable, he will immediately generate code to load its value into a temporary. He thinks this will make his compiler more efficient, because he generates code as soon as possible. Is he right? Why or why not?

4. Give the abstract syntax tree for the following expression (assume the usual order of operations):

```
(a + b) - (c * d)
```

5. Consider a three-address code specification with the following instructions:

- **LOAD(T1) T2** – load from the temporary or variable named T1 (which must be an L-value) and place the value into T2 (which will be an R-value).
- **STORE(T1) T2** – store the value in temporary T1 (which must be an R-value) into the variable T2 (which must be an L-value).
- **OP T1 T2 T3** – perform the operation T3 = T1 OP T2, where T1, T2 and T3 are R-values, and OP is either **ADD**, **MUL** or **SUB**.
Give the generated code for the expression in question 4. Also give the generated code for the following expressions. (If you create temporaries, call them T1, T2, T3, etc.)

(a) \( a + b + c + d \)
(b) \( (a \times b) - c \)
(c) \( a \times (b - c) \)