6 Programming in C and C++ (djg11)

(*a*) You are building a complex-number library where the expression $R = R * A * B$ is coded roughly like

```
struct cpx { double re, im; };
extern cpx A, B; cpx R = { 1.0, 1.0 };
mul(R, R, mul(A, B));
```

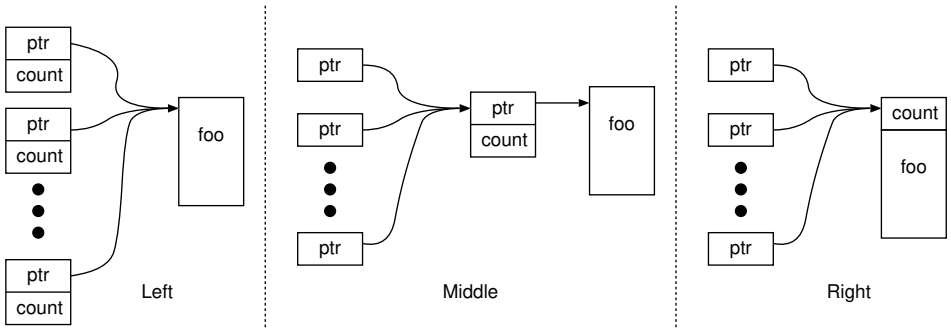
The outer call to `mul` returns its result using pass-by-reference in its first argument: the other two arguments are the operands.

(*i*) Give a complete implementation of the multiply function (and any overloads needed) and an example of using it, using C++ where all arguments to `mul` are references. There might not be any deviations from the original coding style, but if there are, justify changes with a brief comment. [4 marks]

(*ii*) Likewise, give a complete C implementation, including the caller and callee, where all arguments are pointers. To achieve overloading in C, variations on the method name can be used. [4 marks]

(*iii*) Explain how the passing of `R` to `mul` twice (aliasing) can cause an incorrect result in some simple implementations and suggest a fix if either of your implementations might fail. [2 marks]

(*b*) You have no access to the standard library and must code a flexible C++ reference counter class `refct<T>`. It will automatically delete a heap object when its reference count reaches zero.



(*i*) Briefly assess each of the above three design sketches. [2 marks]

(*ii*) Write a C++ implementation of the middle design so that it would behave sensibly under the following (far from ideal) artificial use pattern: [5 marks]

```
foo *copy1 = new foo();  // An object to be managed.
refct<foo> rcf(copy1);   // Add management to the first reference.
foo *copy2 = rcf.new_user(); // Create a second reference to it.
...
rcf.drop();      // Drop one of the references.
rcf.drop();      // Drop the last one, causing foo to be deleted.
```

(*iii*) Criticise the inclusion of the `drop()` method, suggesting an improvement to the overall design that follows the RAII (Resource Acquisition Is Initialization) programming idiom. [3 marks]