```
(((double (double double)) inc) 5)
```

This will apply `inc` $2^4 = 16$ times to 5, yielding 21. This is because `double` is applied **four** *successive* times, since the outer `double` will have an outer pair of doubles, and an inner pair, which will happen again, resulting in 4 nested doubles.

This is **different** to a procedure with three initial nested doubles:
`((double (double (double inc)))) 5)` which would instead produce $5 + 2^3 = 13$.

Here is how applying `double` to `(double double)` yields four nested `double` calls instead of the three above.

```
(((double (double double)) inc) 5)
((((double double) ((double double) x)) inc) 5)
((((double (double *x*)) ((double (double x)) x)) inc) 5)
```

Now substitute the highlighted x.

```
((((double (double ((double (double x)) x))) inc) 5))
```

We now have four nested doubles, so the innermost `double` will produce 2 `inc` calls. The next `double` will turn that into 4, since it will do the procedure that is its argument (which in this case is a procedure to do `inc` twice) twice. The next `double` will thus yield 8 `inc` calls, and the final one will apply 16 `inc` calls, thus resulting in $5 + 16 = 21$.

We can generalize this to say that each outer `double` from this point will result in a doubling of the number of nested doubles. This is because the outermost double will result in `(# of prev doubles (# of prev doubles x))`, which doubles the number of doubles. Due to the nature of `double` all the doubles will eventually be nested, with all the doubles inside that initial nested parenthesis being nested within the outer doubles, and both groups themselves splitting into an outer and inner group however many times necessary to get to the final nesting where each successive `double` call is nested.

Thus, if we added another outer `double` to the original problem, then those four nested doubles would become 8 nested doubles. This means that

```
(((double (double (double double))) inc) 5)
```

will return $5 + 2^8 = 261$ because the outer double will apply the inner `double` (which itself is 4 doubles) twice, resulting in 8 nested `double` calls.

We would then predict

```
(((double (double (double (double double)))) inc) 5)
```

to have 16 nested doubles and return $5 + 2^{16} = 65541$ which it does.