

Implicit Loops





Drahflow

2015-04-14

The bad $1/\infty$

C++:




```
for(int i = 0; i < foo.size(); ++i)
    result[i] = compute(foo[i]);
```

-  `i` repeated 5 times
-  `i` is devoid of meaning
-  `foo.size()` reevaluated
-  `result` better has enough space

The bad $2/\infty$

C++:

```
for(auto &f: foo)
    result.push_back(compute(f));
```

-  `auto` is devoid of meaning
-  `f` is devoid of meaning
-  `push_back` is not very meaningful, either


The bad 3/∞

C++:

```
transform(foo.begin(), foo.end(),  
         back_inserter(result), compute);
```

 `foo` is repeated twice


 `begin/end` just specifies the 95%-case of “all”


 `back_inserter` is not very meaningful, either

The less bad $1/\infty$

Python:

```
[compute(f) for f in foo]
```


 `f` is repeated twice

 only works for arrays, sets

The less bad $2/\infty$

Python:

```
{k: compute(v) for (k, v) in foo.items() }
```


 `k, v` are repeated twice

 `items` is pretty meaningless (and should be `iteritems`)

The kinda ok 1/?

Haskell:

```
map compute foo
```

 works for arrays, sets, dictionaries, user-defined types

APL

J:

compute foo



works for arrays

Function Rank

J:

1 2 3 + 4 5 6

Result:

5 7 9

J: 1 2 3 +/ 4 5 6

Result:

5 6 7

6 7 8

7 8 9



J only allows two arguments per function

Elymas

Elymas:

```
foo compute
```

 works for arrays, dictionaries, functions, user-defined types

Elymas

Elymas:

```
1 [ 1 2 3 ] add
```

Result:

```
[ 2 3 4 ]
```

Elymas:

```
[ 0 1 ] [ 1 2 3 4 ] mul
```

Result:

```
[ 0 2 0 4 ]
```

Elymas

Elymas:

```
{ 1 add } '0.0 ==f
```

```
{ 1 sub } '0.0 ==g
```

```
f g mul =*h
```

```
[ 1 2 3 ] h
```

Result:

```
[ 0 3 8 ]
```