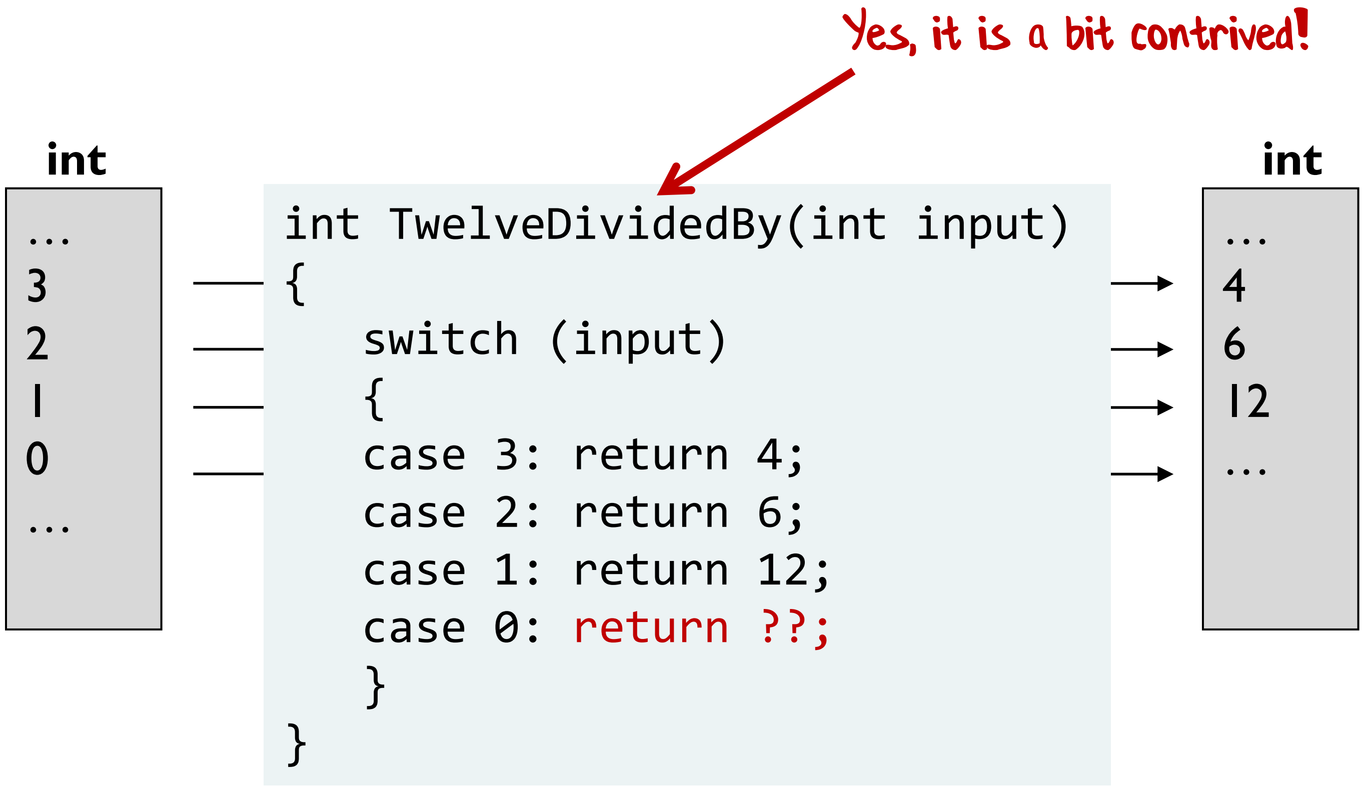
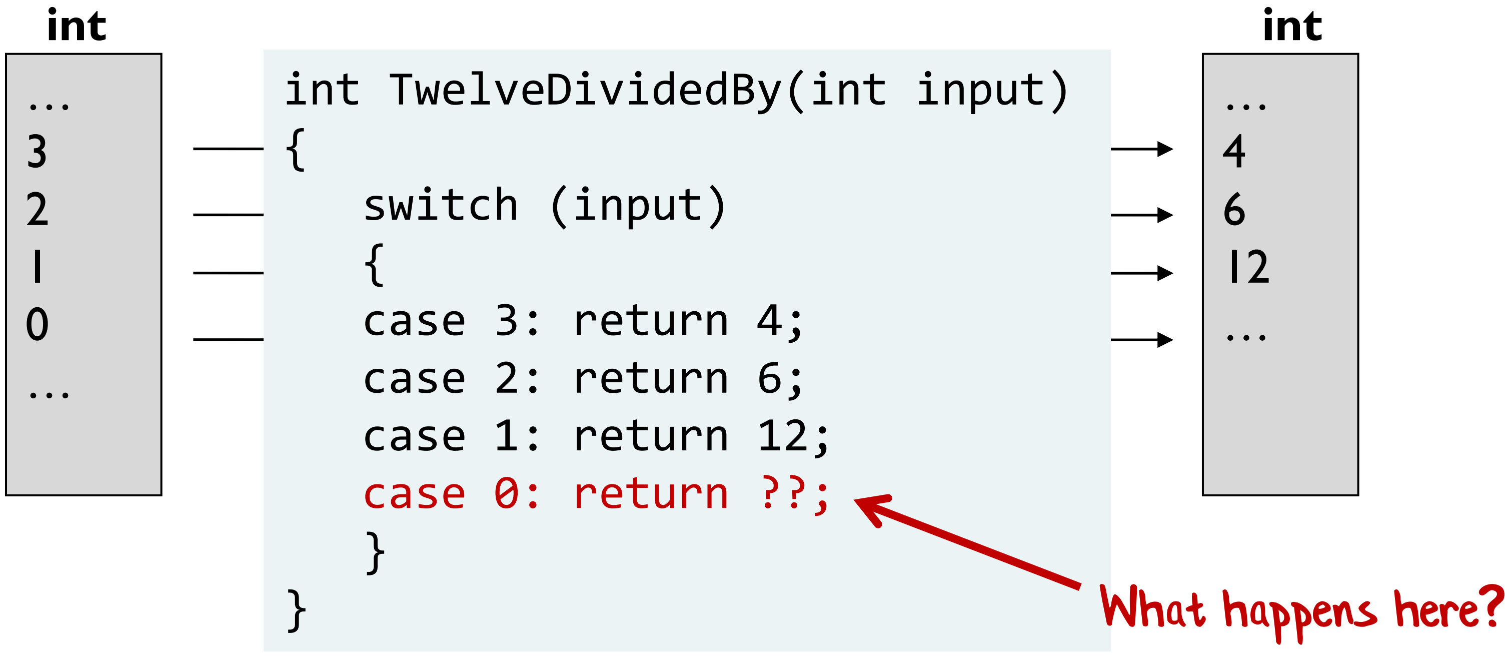
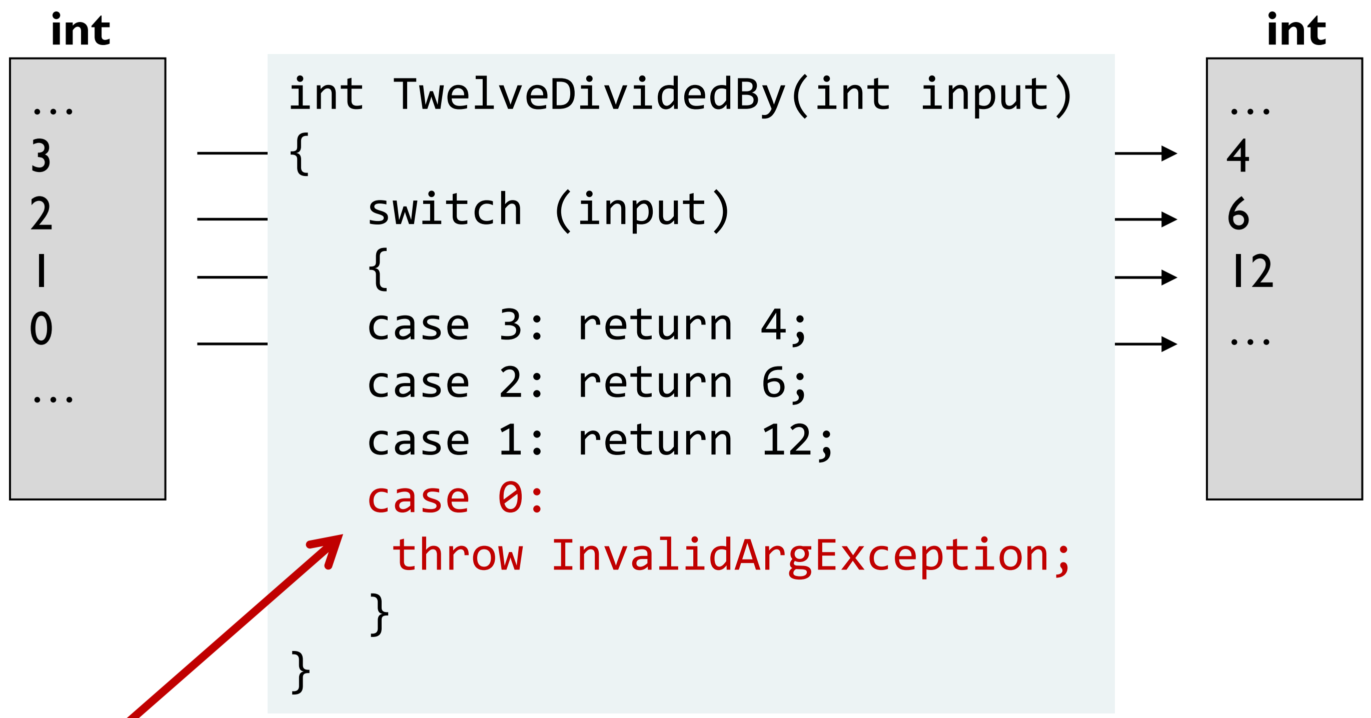
TOTAL FUNCTIONS: MAKING CONTRACTS EXPLICIT



```
int TwelveDividedBy(int input)
{
    switch (input)
    {
    case 3: return 4;
    case 2: return 6;
    case 1: return 12;
    case 0: return ??;
    }
}
```

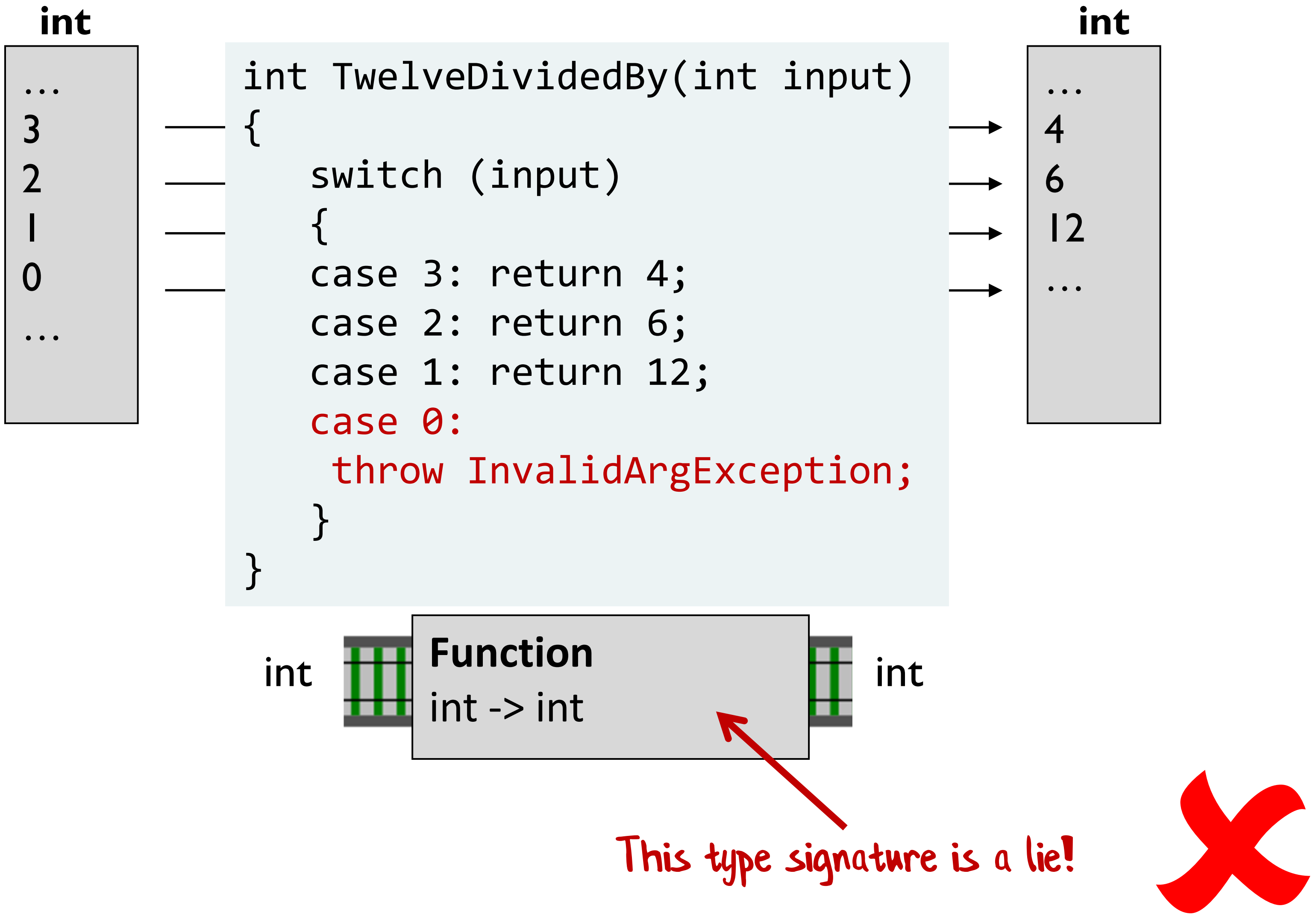


```
int TwelveDividedBy(int input)
{
    switch (input)
    {
    case 3: return 4;
    case 2: return 6;
    case 1: return 12;
    case 0: return ??;
    }
}
```



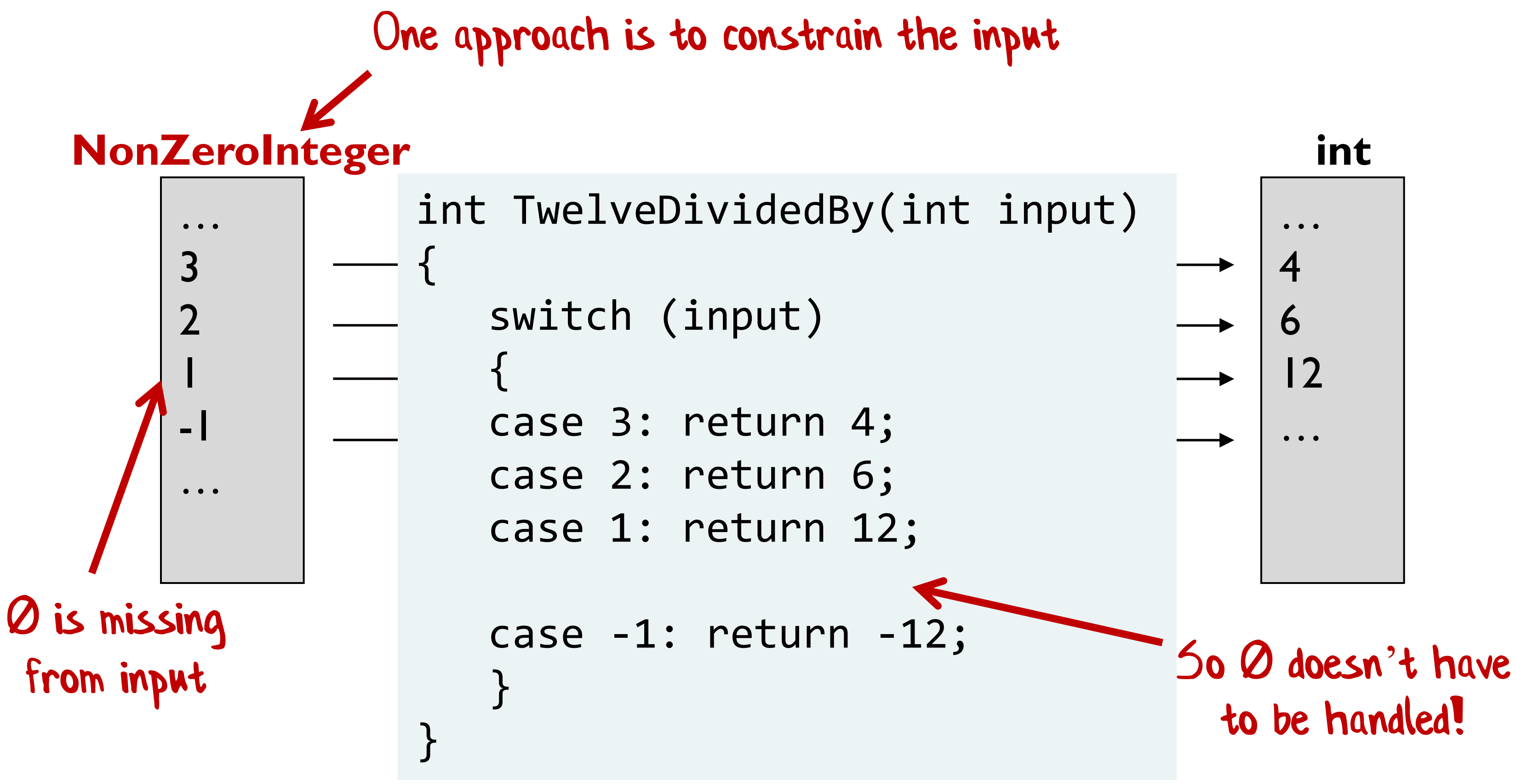
```
int TwelveDividedBy(int input)
{
    switch (input)
    {
    case 3: return 4;
    case 2: return 6;
    case 1: return 12;
    case 0:
     throw InvalidArgException;
    }
}
```

You tell me you can
handle Ø, and then you
complain about it?



```
int TwelveDividedBy(int input)
{
   switch (input)
   {
   case 3: return 4;
   case 2: return 6;
   case 1: return 12;
   case 0:
    throw InvalidArgException;
   }
}
```

But how can we make the contract explicit?

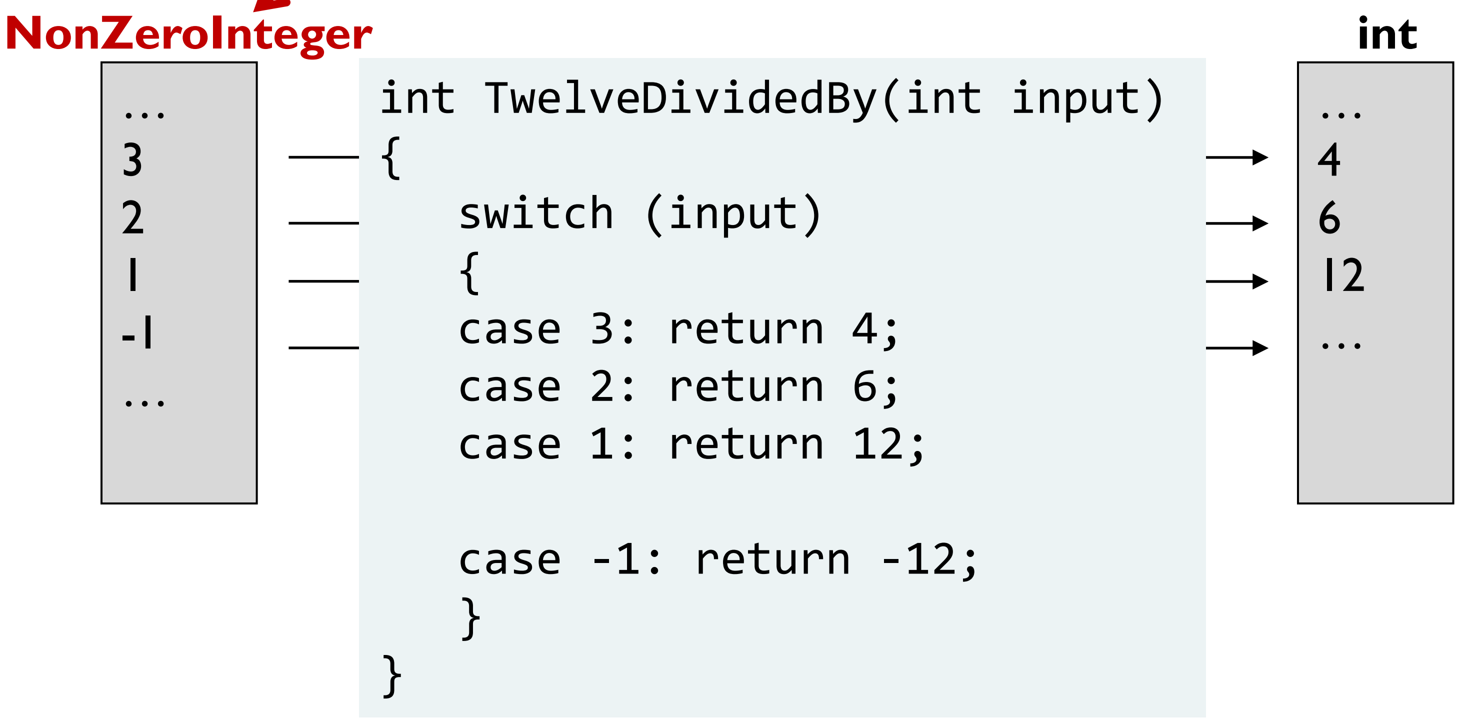


```
int TwelveDividedBy(int input)
{
    switch (input)
    {
    case 3: return 4;
    case 2: return 6;
    case 1: return 12;

    case -1: return -12;
    }
}
```

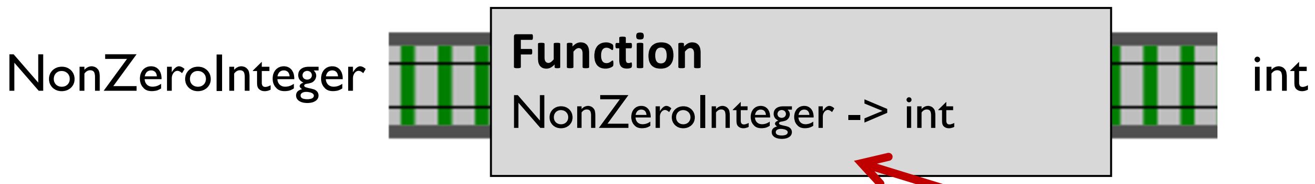
One approach is to constrain the input
NonZeroInteger
...
3
2
1
-1
...
int
...
4
6
12
...
0 is missing from input
So 0 doesn't have to be handled!

One approach is to constrain the input



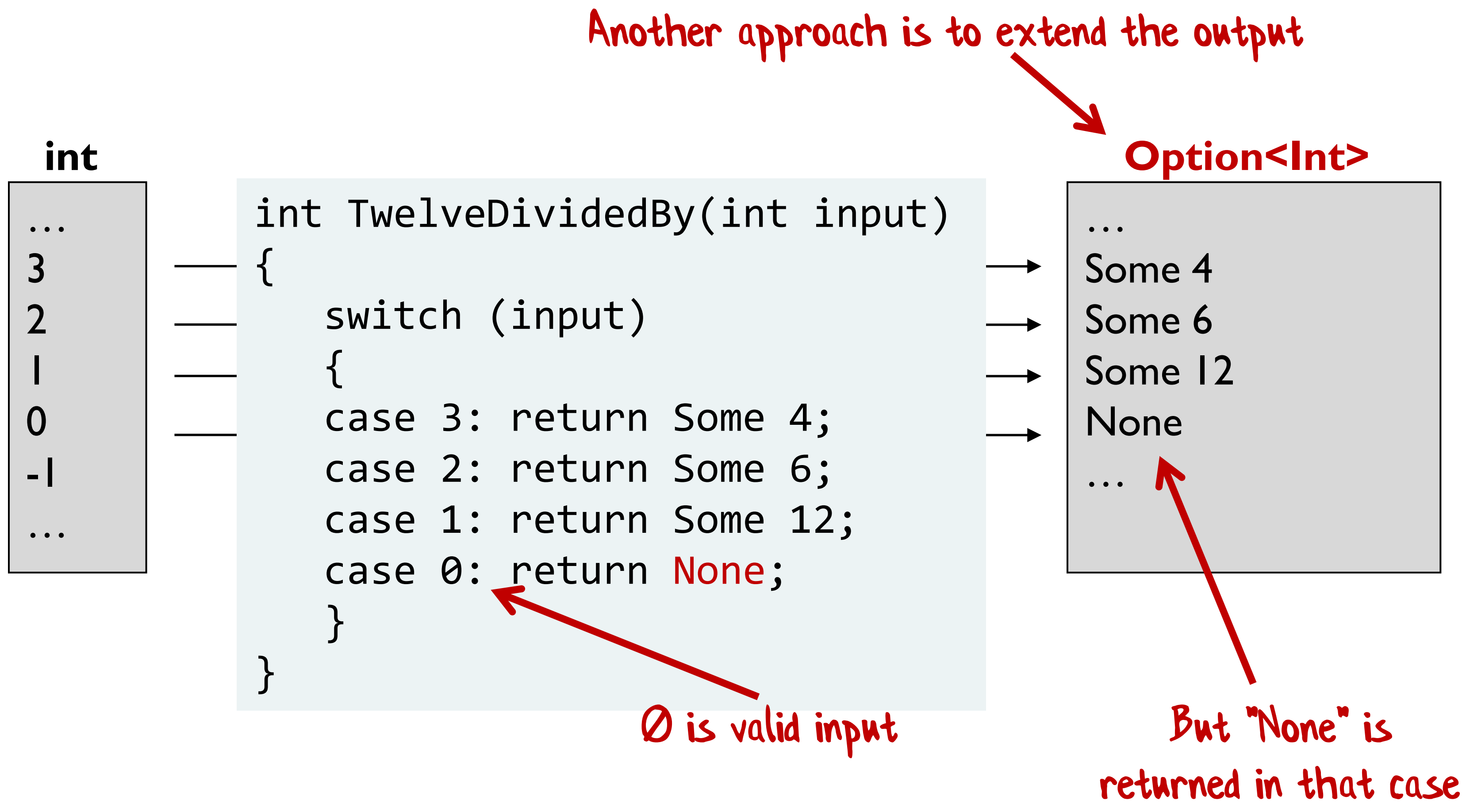
```
int TwelveDividedBy(int input)
{
    switch (input)
    {
    case 3: return 4;
    case 2: return 6;
    case 1: return 12;

    case -1: return -12;
    }
}
```



Contract is explicit.
(Types as documentation!)

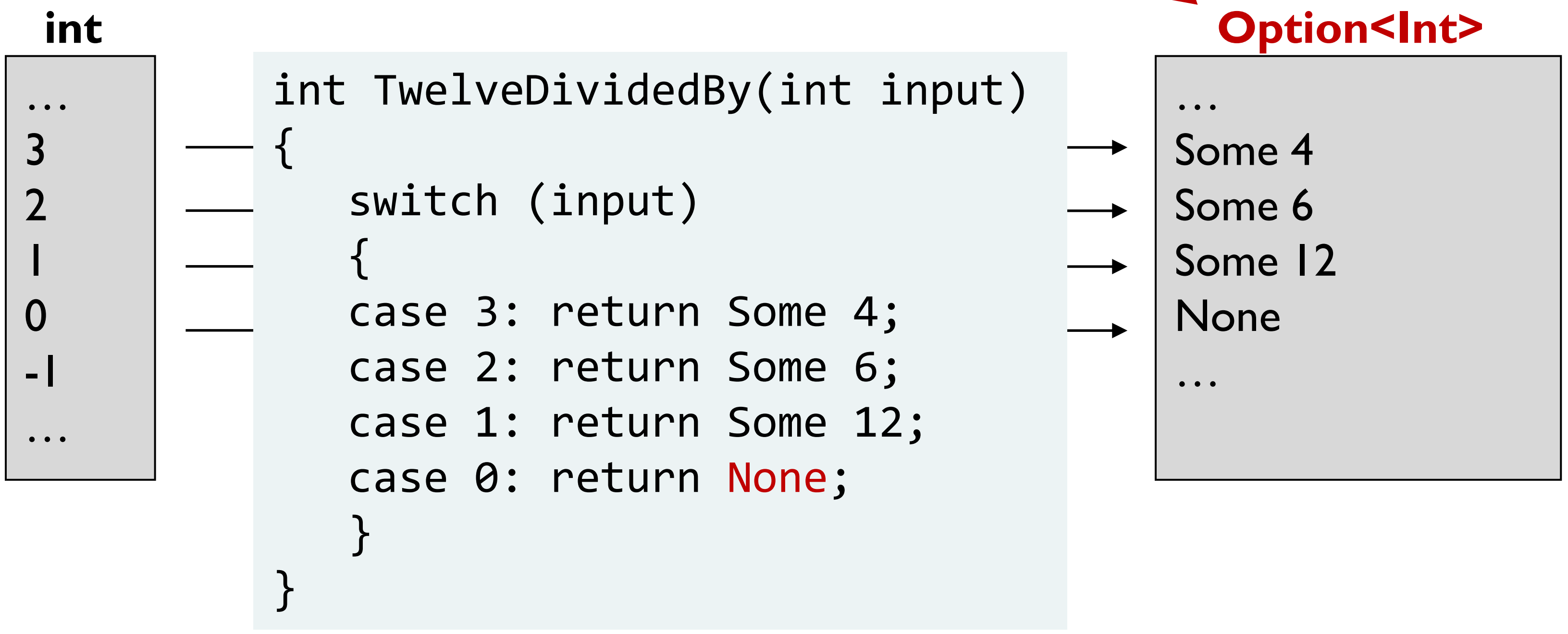
✓



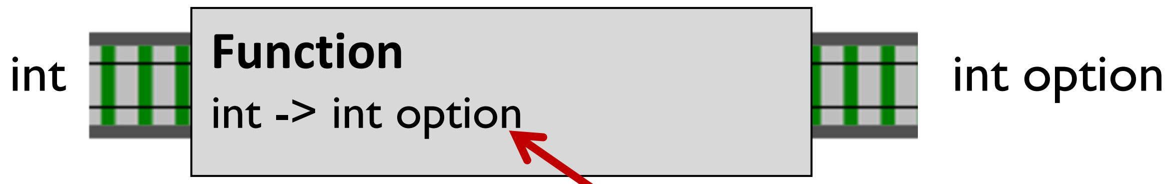
Another approach is to extend the output
int
...
3
2
1
0
-1
...
Option<Int>
...
Some 4
Some 6
Some 12
None
...
0 is valid input
But "None" is returned in that case

```
int TwelveDividedBy(int input)
{
   switch (input)
   {
   case 3: return Some 4;
   case 2: return Some 6;
   case 1: return Some 12;
   case 0: return None;
   }
}
```

Another approach is to extend the output



```
int TwelveDividedBy(int input)
{
    switch (input)
    {
    case 3: return Some 4;
    case 2: return Some 6;
    case 1: return Some 12;
    case 0: return None;
    }
}
```



Contract is explicit.
(Types as documentation!)

Explicit contracts mean fewer bugs

int -> int

Could cause runtime error if misused.

NonZeroInteger -> int

int -> int option

These can NEVER cause runtime errors.

Demo:

`07a-TotalFunctions.fsx`

Exercise:

`07b-Exercise-TotalFunctions.fsx`