Em Haskell:

```haskell
add :: Int -> Int -> Int
add a b = a + b

main :: IO ()
main = do
    let r = add 1 2

    putStrLn $ show r
```

Em C++:

```cpp
template <int a, int b> struct add {
    static constexpr int value = a + b;
};

int main() {
    auto r = add<1,2>::value;

    std::cout << r << std::endl;
    return 0;
}
```

Note-se que imprimir um valor é um *side-effect* mas a função de cálculo do resultado é pura.

Em C++ o valor de r será calculado em tempo de compilação.

```haskell
addL:: [Int] -> Int
addL [] = 0
addL (h:t) = h + addL t
```

```cpp
template<typename T> T addL(vector<T> v){
    return accumulate(v.begin(), v.end(), 0, add<T>);
}
```

```haskell
max :: (Num a, Ord a) => a -> a -> a
max a b | a > b = a
        | otherwise = b
```

```cpp
template<typename T> T max(T x, T y){
    return x > y? x : y;
}
```

```haskell
filter' :: (a -> Bool) -> [a] -> [a]
filter' f [] = []
filter' f (h:t)  | f h = h: filter' f t
                 | otherwise = filter' f t
```

```
template<typename T, typename P> vector<T> filter(vector<T> v,P f){
    vector<T> res;
    copy_if(v.begin(),v.end(),back_inserter(res),f);
    return res;
}
```

```
map :: (a -> b) -> [a] -> [b]
map f [] = []
map f (h:t) = f h : map f t
```

```
ver erro
```

```
foldl :: Foldable t => (b -> a -> b) -> b -> t a -> b
foldl f b [] = b
foldl f b (h:t) = foldl f (f b h) t
```

```
template<typename A, typename B, typename F> B foldl(F f, B b, vector<A> v){
    return accumulate(v.begin(), v.end(), b, f);
}
```

```
reverse :: [a] -> [a]
reverse = foldl (flip (:)) []
```

```
template<typename T> vector<T> my_reverse(vector<T> v){
    return foldl([] (vector<T> vec, T t){ vec.insert(vec.begin(),t); return vec;},vector<T>(
}
```