

Say Goodbye to the For Loop with High Order Functions

Mike Harris
@MikeMKH

Pop Quiz

```
main ( )
{
    int i, j, k, l;
    float x[8][2][8][2];

    for (i = 0; i < 8; i++)
        for (j = i; j < 8; j++)
            for (k = 0; k < 2; k++)
                for (l = 0; l < 2; l++)
                {
                    if ((i == j) && (k == l))
                        x[i][k][j][l] = 0.8;
                    else
                        x[i][k][j][l] = 0.8;
                    if (x[i][k][j][l] < 0.0)
                        abort ();
                }

    exit (0);
}
```

```
int f(void)
{
    static _Complex double t;
    int i, j;
    for(i = 0; i < 2; i++)
        for(j = 0; j < 2; j++)
            t = .5 * 1.0;
    return t;
}
```

```
int n;

void foo (int i)
{
    int a, b;

    if (!i)
        for (a = 1; a < 4; a++)
            if (a)
                for (b = 1; b < 3; b++)
                    foo (b);

    n++;
}
```

I Have No Idea Either

Agenda

- Problem
- Map
- Filter
- Fold
- Theory

Problem Statment

Realistic Data

```
IImmutableList<(int zip, double price, int quantity)> orders =  
    ImmutableList.Create(  
        (53202, 1.89, 3),  
        (60191, 1.99, 2),  
        (60060, 0.99, 7),  
        (53202, 1.29, 8),  
        (60191, 1.89, 2),  
        (53202, 0.99, 3)  
    );
```

Find the total for 53202?

For Loop

```
var total = 0.0;
for (int i = 0; i < orders.Count(); i++)
{
    if (orders[i].zip == 53202)
        total += orders[i].price * orders[i].quantity;
}
```

Foreach Loop

```
var total = 0.0;
foreach (var order in orders)
{
    if (order.zip == 53202)
        total += order.price * order.quantity;
}
```

Higher Order Function

```
var total = orders
    .Where(order => order.zip == 53202)
    .Select(order => order.price * order.quantity)
    .Aggregate(0.0, (sub, amount) => sub + amount);
```

Higher Order Functions

Map

$(a \rightarrow b) \rightarrow [a] \rightarrow [b]$

```
private sealed class SelectArrayIterator<TSource, TResult> : Iterator<TResult>, IPartition<TResult>
{
    private readonly TSource[] _source;
    private readonly Func<TSource, TResult> _selector;

    public override bool MoveNext()
    {
        if (_state < 1 | _state == _source.Length + 1)
        {
            Dispose();
            return false;
        }

        int index = _state++ - 1;
        _current = _selector(_source[index]);
        return true;
    }
}
```



```
let map mapping x =  
  match x with  
  | [] -> []  
  | [h] -> [mapping h]  
  | h::t ->  
    let cons = freshConsNoTail (mapping h)  
    mapToFreshConsTail cons mapping t  
    cons
```

Filter

$(a \rightarrow \text{bool}) \rightarrow [a] \rightarrow [a]$

```
internal sealed class WhereArrayIterator<TSource> : Iterator<TSource>, IListProvider<TSource>
{
    private readonly TSource[] _source;
    private readonly Func<TSource, bool> _predicate;

    public override bool MoveNext()
    {
        int index = _state - 1;
        TSource[] source = _source;

        while (unchecked((uint)index < (uint)source.Length))
        {
            TSource item = source[index];
            index = _state++;
            if (_predicate(item))
            {
                _current = item;
                return true;
            }
        }

        Dispose();
        return false;
    }
}
```

```
let rec filter predicate l =  
  match l with  
  | [] -> l  
  | h :: ([] as nil) -> if predicate h then l else nil  
  | h::t ->  
    if predicate h then  
      let cons = freshConsNoTail h  
      filterToFreshConsTail cons predicate t  
      cons  
    else  
      filter predicate t
```

Fold

(state -> a -> state) -> state -> [a] -> state

```
public static TAccumulate Aggregate<TSource, TAccumulate>(this IEnumerable<TSource> source,
TAccumulate seed, Func<TAccumulate, TSource, TAccumulate> func)
{
    if (source == null)
    {
        throw Error.ArgumentNull(nameof(source));
    }

    if (func == null)
    {
        throw Error.ArgumentNull(nameof(func));
    }

    TAccumulate result = seed;
    foreach (TSource element in source)
    {
        result = func(result, element);
    }

    return result;
}
```

```
let fold<'T, 'State> folder (state: 'State) (list: 'T list) =  
    match list with  
    | [] -> state  
    | _ ->  
        let f = OptimizedClosures.FSharpFunc<_, _, _>.Adapt(folder)  
        let mutable acc = state  
        for x in list do  
            acc <- f.Invoke(acc, x)  
        acc
```

Theory

Thank you

Next Steps

Images

gcc Source Code

- example 1, <https://github.com/gcc-mirror/gcc/blob/e11be3ea01eaf8acd8cd86d3f9c427621b64e6b4/gcc/testsuite/gcc.c-torture/execute/930614-2.c#L1-L20>
- example 2, <https://github.com/gcc-mirror/gcc/blob/e11be3ea01eaf8acd8cd86d3f9c427621b64e6b4/gcc/testsuite/gcc.c-torture/compile/pr25513.c#L1-L9>
- example 3, <https://github.com/gcc-mirror/gcc/blob/e11be3ea01eaf8acd8cd86d3f9c427621b64e6b4/gcc/testsuite/gcc.c-torture/compile/pr43186.c#L1-L15>

LINQ Source Code

- Select, <https://github.com/dotnet/corefx/blob/a673a117846205fc1a5c648c29451ff3da83554d/src/System.Linq/src/System.Linq/Select.cs#L199-L226>
- Where, <https://github.com/dotnet/corefx/blob/a673a117846205fc1a5c648c29451ff3da83554d/src/System.Linq/src/System.Linq/Where.cs#L198-L255>
- Aggregate, <https://github.com/dotnet/corefx/blob/a673a117846205fc1a5c648c29451ff3da83554d/src/System.Linq/src/System.Linq/Aggregate.cs#L40-L59>

F# Source Code

- map, <https://github.com/Microsoft/visualfsharp/blob/f62158bae5a300be60abf3d97ae7cb4f83e7267d/src/fsharp/FSharp.Core/local.fs#L247-L254>
- filter, <https://github.com/Microsoft/visualfsharp/blob/f62158bae5a300be60abf3d97ae7cb4f83e7267d/src/fsharp/FSharp.Core/local.fs#L490-L500>
- fold, <https://github.com/fsharp/fsharp/blob/e19ddca7d6049ae04cc6a827e803555285d19b26/src/fsharp/FSharp.Core/list.fs#L214-L222>