UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical &
Computer Engineering

ECE 150 *Fundamentals of Programming*

Reduce

Douglas Wilhelm Harder, M.Math.
Prof. Hiren Patel, Ph.D.
hiren.patel@uwaterloo.ca dwharder@uwaterloo.ca

Outline

- In this lesson, we will:
  - Define a reduction of an array
  - Look at implementations of various reductions
  - See how to extract commonality
  - Introduce a reduce function
  - Look at the implementation and examples

Introduction

- We have already discussed algorithms such as:
  - Finding the maximum entry of an array
  - Finding the sum of the entries of an array

- These are collectively called *reductions* as they *reduce* the information in the array to a single value
  - A single piece of information

- Each reduction we have implemented so far
  has had its own implementation

Introduction

- Question:
  - Can we find common features between these reductions?

```
void max( double array[], std::size_t capacity ) {
    double max_val{ array[0] };

    for ( std::size_t k{1}; k < capacity; ++k ) {
        if ( array[k] > max_val ) {
            max_val = array[k];
        }
    }

    return max_val;
}
```

```
void sum( double array[], std::size_t capacity ) {
    double array_sum{ 0.0 };

    for ( std::size_t k{0}; k < capacity; ++k ) {
        array_sum += array[k];
    }

    return array_sum;
}
```

Introduction

- Question:
  - Can we at least get the index starting at `0`?

```
void max( double array[], std::size_t capacity ) {
    double max_val{ -std::numeric_limits<double>::infinity() };

    for ( std::size_t k{0}; k < capacity; ++k ) {
        if ( array[k] > max_val ) {
            max_val = array[k];
        }
    }

    return max_val;
}
```

```
void sum( double array[], std::size_t capacity ) {
    double array_sum{ 0.0 };

    for ( std::size_t k{0}; k < capacity; ++k ) {
        array_sum += array[k];
    }

    return array_sum;
}
```

Introduction

- Question:
  - Can we simplify the function bodies?
  - In both cases, the updated value is a function of the value and `array[k]`

```
void max( double array[], std::size_t capacity ) {
    double max_val{ -std::numeric_limits<double>::infinity() };

    for ( std::size_t k{0}; k < capacity; ++k ) {
        max_val = std::max( max_val, array[k] );
    }

    return max_val;
}
```

```
void sum( double array[], std::size_t capacity ) {
    double array_sum{ 0.0 };

    for ( std::size_t k{0}; k < capacity; ++k ) {
        array_sum = array_sum + array[k];
    }

    return array_sum;
}
```

Initial implementation

- Could we not get the user to pass:
  - The initial value?
  - The bivariate function that performs the reduction?

- The function declaration would be:

```
double reduce( double      array[],
               std::size_t capacity,
               double      x0,
               std::function<double(double, double)> accumulator );
```

  - We call the function an *accumulator*,
    as it accumulates the information about the array

Initial implementation

- The implementation is also straightforward:

```
double reduce( double      array[],
               std::size_t capacity,
               double      x0,
               std::function<double(double, double)> accumulator ) {
    double result{ x0 };

    for ( std::size_t k{0}; k < capacity; ++k ) {
        result = accumulator( result, array[k] );
    }

    return result;
}
```

Generalizing the range

- Have the algorithm work from

```
array[begin], ... , array[end - 1]
```

- This is rather easy, too:

```
double reduce( double      array[],
               std::size_t begin,
               std::size_t end,
               double      x0,
               std::function<double(double, double)> accumulator ) {
    double result{ x0 };

    for ( std::size_t k{begin}; k < end; ++k ) {
        result = accumulator( result, array[k] );
    }

    return result;
}
```

Example 1

- What does this code do?

```
int main() {
    std::size_t N{ 10 };
    double data{ 3.2, -5.4,  1.9,  8.6,  0.7,
                 6.5,  2.0,  7.1, -4.3, -9.8 };

    std::cout << reduce( data, 0, N, 1.0, product ) << std::endl;

    return 0;
}

double product( double x, double y ) {
    return x*y;
}
```

Example 2

- What does this code do?

```
int main() {
    std::size_t N{ 10 };
    double data{ 3.2, -5.4,  1.9,  8.6,  0.7,
                 6.5,  2.0,  7.1, -4.3, -9.8 };

    std::cout << reduce( data, 0, N,
                         std::numeric_limits<double>::infinity(), min )
              << std::endl;

    return 0;
}

double min( double x, double y ) {
    if ( x <= y ) {
        return x;
    } else {
        return y;
    }
}
```

The standard library

- In the standard library, there is a

  `std::reduce(…)`

  in the header

  `#include <numeric>`

  - Again, despite it appearing there are many function evaluations,
    a good compiler will eliminate these and simply inline these operations
  - Rather than passing an array pointer and indices,
    you pass the addresses of `array[begin]` and `array[end]`

The standard library

- The functions assumes that the accumulator is both commutative and associative, so

```
               accumulator(x, y)  == accumulator(y, x)
accumulator(x, accumulator(y, z)) == accumulator( accumulator(x, y), z)
```

  – For example:

```
      (x + y) == (y + x)
(x + (y + z)) == ((x + y) + z)

      max(x, y) == max(y, x)
max(x, max(y, z)) == max(max(x, y), z)
```

  – This allows the implementation to be parallelized

Summary

- Following this lesson, you now:
  - Understand what a reduction is
  - Seen how to implement a reduction
  - Looked at calculating the sum, minimum and maximum of an array

References

[1] https://en.cppreference.com/w/cpp/algorithm/reduce

Colophon

These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`.

The photographs of lilacs in bloom appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens on May 27, 2018 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.

Disclaimer