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

ECE 150 *Fundamentals of Programming*

Transform-reduce

ECE150

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

Outline

- In this lesson, we will:
  - Define the transformation-reduction of an array
  - Look at implementations of various reductions
  - Introduce a transform-reduce function
  - Look at the implementation and examples

Introduction

- We have looked at reductions
  - We will now look at other reductions that suggest we need more

Introduction

- Suppose we want to find the maximum absolute value
  - We would need a new accumulator:

```
void max_abs( double x, double y ) {
    x = std::abs( x );
    y = std::abs( y );

    if ( x >= y ) {
        return x;
    } else {
        return y;
    }
}
```

Introduction

- Suppose we want to find the sum of the squares:
  - We would need a new accumulator:

```
void sum_of_squares( double x, double y ) {
    return x + y*y;
}
```

  - Issue: this accumulator is neither commutative nor associative
    - The implementation is no longer parallelizable…

Introduction

- Instead, we have

```
max_val = std::max( max_val, std::abs( array[k] ) );
array_sum = array_sum + array[k]*array[k];
```

- Instead, we use the commutative and associative functions,
  but we need to provide a function that *transforms* the array entry

```
max_val = std::max( max_val, T( array[k] ) );
array_sum = array_sum + T( array[k] );
```

Generalizing the range

- Thus, we have:

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

    for ( std::size_t k{begin}; k < end; ++k ) {
        result = accumulator( result, transform( 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 << transform_reduce( data, 0, N,
                                   -std::numeric_limits<double>::infinity(),
                                    max, abs_val )
              << std::endl;

    return 0;
}

double max( double x, double y ) {
    if ( x >= y ) {
        return x;
    } else {
        return y;
    }
}

double abs_val( double x ) {
    if ( x >= 0 ) {
        return x;
    } else {
        return -x;
    }
}
```

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 << transform_reduce( data, 0, N, 0.0, sum, sqr )
              << std::endl;

    return 0;
}

double sum( double x, double y ) {
    return x + y;
}

double sqr( double x ) {
    return x*x;
}
```

The standard library

- In the standard library, there is a

  `std::transform_reduce(…)`

  in the header

  `#include <numeric>`

  – Remember, rather than passing an array pointer and indices, you pass the addresses of `array[begin]` and `array[end]`

Summary

- Following this lesson, you now:
  - Understand what a transformation-reduction is
  - Seen how to implement this function
  - Looked at calculating the sum of squares of the entries of the array, and the maximum entry in absolute value
  - A fun game is to determine how many algorithms can be implemented using a transform-reduce

References

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

These slides are provided for the ECE 150 *Fundamentals of Programming* course taught at the University of Waterloo. The material in it reflects the authors' best judgment in light of the information available to them at the time of preparation. Any reliance on these course slides by any party for any other purpose are the responsibility of such parties. The authors accept no responsibility for damages, if any, suffered by any party as a result of decisions made or actions based on these course slides for any other purpose than that for which it was intended.