Outline

- In this lesson, we will:
  - Examine and implement recursive mathematical formulas:
    - The factorial function
    - Binomial coefficients
    - The Fibonacci numbers
    - The calculation of x^n for an integer exponent

Factorial

- The factorial function can be defined recursively:

$$n! = \begin{cases} 1 & n \leq 1 \\ n(n-1)! & n \geq 2 \end{cases}$$

- Such a recursive definition is easy to implement:

```
unsigned int factorial( unsigned int n ) {
    if ( n <= 1 ) {
        return 1;
    } else {
      unsigned int simpler_result{ factorial( n - 1 ) };
      return n * simpler_result;
    }
}               return n * factorial( n - 1 );
```


Binomial coefficients

- Binomial coefficients can also be defined recursively:

$$\binom{n}{k} = \begin{cases} 0 & k > n \\ 1 & k = 0 \text{ or } k = n \\ \binom{n-1}{k} + \binom{n-1}{k-1} & \text{otherwise} \end{cases}$$

- This is also easy to implement:

```
unsigned int binomial( unsigned int n, unsigned int k ) {
    if ( k > n ) {
        return 0;
    } else if ( (k == 0) || (k == n) ) {
        return 1;
    } else {
        unsigned int result_1{ binomial( n - 1, k ) };
        unsigned int result_2{ binomial( n - 1, k - 1 ) };
        return result_1 + result_2;
    }
}
 return binomial( n - 1, k ) + binomial( n - 1, k - 1 );
```

Fibonacci numbers

- The Fibonacci numbers are defined recursively:

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F(n-1) + F(n-2) & \text{otherwise} \end{cases}$$

- This is also easy to implement:

```
unsigned int fibonacci( unsigned int n ) {
    if ( n == 0 ) {
        return 0;
    } else if ( n == 1 ) {
        return 1;
    } else {
        unsigned int result_1{ fibonacci( n - 1 ) };
        unsigned int result_2{ fibonacci( n - 2 ) };
        return result_1 + result_2;
    }
}
```

```
return fibonacci( n - 1 ) + fibonacci( n - 2 );
```


Fibonacci numbers

- Here is a different implementation of the Fibonacci numbers:

```
unsigned int fibonacci( unsigned int n ) {
    unsigned int values[2]{0, 1};

    for ( unsigned int k{2}; k <= n; ++k ) {
        values[k%2] = values[0] + values[1];
    }

    return values[n%2];
}
```

144
89

This is called an *iterative* implementation
- The statements in the for-loop body are *iterated* $n - 1$ times

Fibonacci numbers

- If we try calculating the Fibonacci numbers from $F(0)$ to $F(47)$, here are the following times:

Implementation	**Time (s)**
Recursive implementation	109.437
Iterative implementation	0.015

- In your course on algorithms, you will learn about *dynamic programming* or *memoization*
  - This algorithm design technique can significantly improve the run-time of recursive implementations

Fibonacci numbers

- Mathematicians have come up with an alternative recursive definition of the Fibonacci numbers:

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \text{ or } n = 2 \\ F(k)\left(2F(k+1) - F(k)\right) & \text{if } n = 2k\text{; that is, } n \text{ is even} \\ F(k+1)^2 - F(k)^2 & \text{if } n = 2k+1\text{; that is, } n \text{ is odd} \end{cases}$$

  - Exercise: implement this variation yourself

Integer exponents

- Here is a straight-forward recursive definition of calculating x^n

$$x^n = \begin{cases} 1 & n = 0 \\ \dfrac{1}{x^{-n}} & n < 0 \\ x\left(x^{n-1}\right) & \text{otherwise} \end{cases}$$

- This is also easy to implement:

```
double power( double x, int n ) {
    if ( n == 0 ) {
        return 1.0;
    } else if ( n < 0 ) {
        return 1.0/power( x, -n );
    } else {
        double result{ power( x, n - 1 ) };
        return x*result;
    }
}
```


Integer exponents

- Consider this alternative recursive definition of calculating x^n

$$x^n = \begin{cases} 1 & n = 0 \\ \dfrac{1}{x^{-n}} & n < 0 \\ \left(x^k\right)^2 & \text{if } n = 2k\text{; that is, } n \text{ is even} \\ x\left(x^k\right)^2 & \text{if } n = 2k+1\text{; that is, } n \text{ is odd} \end{cases}$$

- This is also easy to implement:

```
double power( double x, int n ) {
    if ( n == 0 ) {
        return 1.0;
    } else if ( n < 0 ) {
        return 1.0/power( x, -n );
    } else if ( (n%2) == 0 ) {
        double result{ power( x, n/2 ) };
        return result*result;
    } else {
        double result{ power( x, n/2 ) };
        return x*result*result;
    }
}
```

Remember, we are using integer division here

Integer exponents

- Question: What happens if I implement the following?

```
double power( double x, int n ) {
    if ( n == 0 ) {
        return 1.0;
    } else if ( n < 0 ) {
        return 1.0/power( x, -n );
    } else if ( (n%2) == 0 ) {
        return power( x, n/2 )*power( x, n/2 );
    } else {
        return x*power( x, n/2 )*power( x, n/2 );
    }
}
```


Summary

- Following this presentation, you now:
  - Understand the recursive implementation of:
    - The factorial function
    - The binomial coefficients
  - Understand that the naïve recursive definition of the Fibonacci numbers translates poorly to an implementation
    - The iterative variation is much more efficient
  - Have been exposed to a much more efficient recursive definition
  - Realize there are often many different recursive definitions, as seen with the calculation of x^n
  - Understand that some implementations can be much more efficient

References

[1] Wikipedia,
https://en.wikipedia.org/wiki/Factorial
https://en.wikipedia.org/wiki/Binomial_coefficient
https://en.wikipedia.org/wiki/Fibonacci_numbers
https://en.wikipedia.org/wiki/Exponentiation#Integer_exponents

Acknowledgments

None so far.

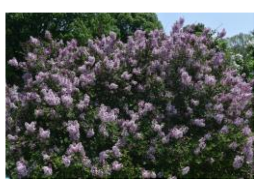
Colophon

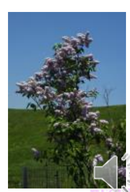
These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`.

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.