

# Functions in C++

# Outline for Today

- ***Functions in C++***
  - How C++ organizes code.
- ***Some Simple Functions***
  - Getting comfortable with the language.
- ***Intro to Recursion***
  - A new perspective on problem-solving.

# Writing Functions in C++

# A Simple C++ Program

Hip hip, hooray!

Hip hip, hooray!  
Hip hip, hooray!  
Hip hip, hooray!

What Went Wrong?

# One-Pass Compilation

- When you compile a C++ program, the compiler reads your code from top to bottom.
- If you call a function that you haven't yet written, the compiler will get Very Upset and will say mean things to you.
- You will absolutely encounter this issue. How do you address this?





# Option 1: Reorder Your Functions

Option 2: Use Forward Declarations

# Forward Declarations

- A ***forward declaration*** is a statement that tells the C++ compiler about an upcoming function.
  - The textbook calls these ***function prototypes***. It's different names for the same thing.
- Forward declarations look like this:  
***return-type function-name(parameters);***
- Essentially, start off like you're defining the function as usual, but put a semicolon instead of the function body.
- Once the compiler has seen a forward declaration, you can go and call that function as normal.

# Some More Functions

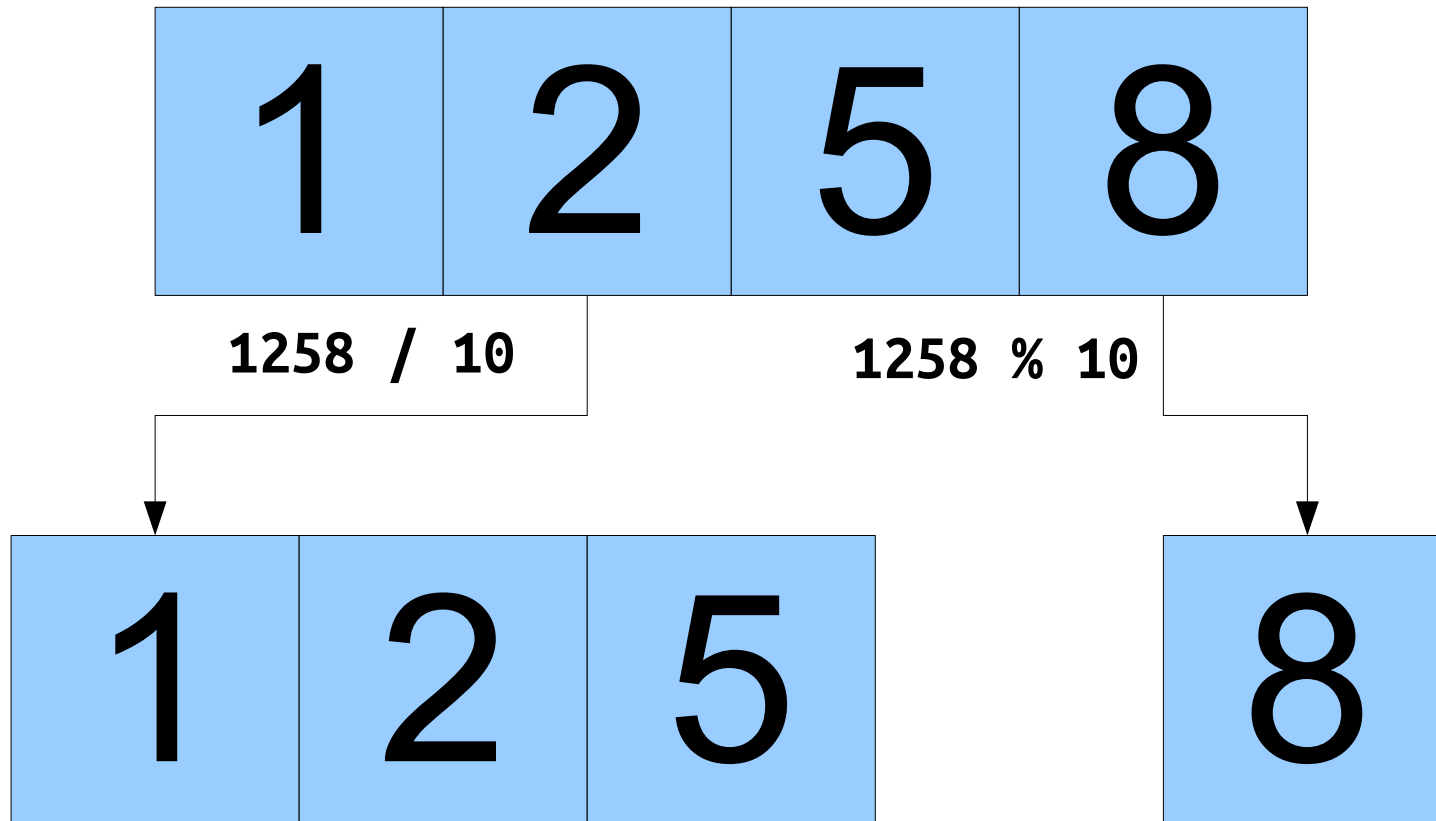
# Summing Up Digits

- Ever seen that test for divisibility by three?  
*Add the digits of the number; if the sum is divisible by three, the original number is divisible by three (and vice-versa).*
- Let's write a function

```
int sumOfDigitsOf(int n)
```

that takes in a number and returns the sum of its digits.

# Working One Digit at a Time



Dividing two integers in C++ ***always*** produces an integer by dropping any decimal value. Check the textbook for how to override this behavior.

# Functions in Action

```
int main() {  
    int n = getInteger("Enter an integer: ");  
    int digitSum = sumOfDigitsOf(n);  
    cout << n << " sums to " << digitSum << endl;  
  
    return 0;  
}
```

# Functions in Action

```
int main() {  
    int n = getInteger("Enter an integer: ");  
    int digitSum = sumOfDigitsOf(n);  
    cout << n << " sums to " << digitSum << endl;  
  
    return 0;  
}
```



# Functions in Action

```
int main() {  
    int n = getInteger("Enter an integer: ");  
    int digitSum = sumOfDigitsOf(n);  
    cout << n << " sums to " << digitSum << endl;  
  
    return 0;  
}
```

137

int n

# Functions in Action

```
int main() {  
    int n = getInteger("Enter an integer: ");  
    int digitSum = sumOfDigitsOf(n);  
    cout << n << " sums to " << digitSum << endl;  
  
    return 0;  
}
```

137

int n

The variable `n` actually is an honest-to-goodness integer, not a pointer to an integer that lives somewhere else. In C++, all variables stand for actual objects unless stated otherwise. (More on that later.)

# Functions in Action

```
int main() {  
    int n = getInteger("Enter an integer: ");  
    int digitSum = sumOfDigitsOf(n);  
    cout << n << " sums to " << digitSum << endl;  
  
    return 0;  
}
```

137  
int n

# Functions in Action

```
int main() {  
    int n = getInteger("Enter an integer: ");  
    int digitSum = sumOfDigitsOf(n);  
    cout << n << " sums to " << digitSum << endl;  
  
    return 0;  
}
```

137

int n

# Functions in Action

```
int sumOfDigitsOf(int n) {  
    int result = 0;  
    while (n > 0) {  
        result += (n % 10);  
        n /= 10;  
    }  
    return result;  
}
```

137  
int n



When we call `sumOfDigitsOf`, we get our own variable named `n`. It's separate from the variable `n` in `main()`, and changes to this variable `n` don't reflect back in `main`.

# Functions in Action

```
int sumOfDigitsOf(int n) {  
    int result = 0;  
    while (n > 0) {  
        result += (n % 10);  
        n /= 10;  
    }  
    return result;  
}
```

137

int n

# Functions in Action

```
int sumOfDigitsOf(int n) {  
    int result = 0;  
    while (n > 0) {  
        result += (n % 10);  
        n /= 10;  
    }  
    return result;  
}
```

137

int n

0

int result

# Functions in Action

```
int sumOfDigitsOf(int n) {  
    int result = 0;  
    while (n > 0) {  
        result += (n % 10);  
        n /= 10;  
    }  
    return result;  
}
```

137

int n

0

int result



# Functions in Action

```
int sumOfDigitsOf(int n) {  
    int result = 0;  
    while (n > 0) {  
        result += (n % 10);  
        n /= 10;  
    }  
    return result;  
}
```

137

int n

0

int result

# Functions in Action

```
int sumOfDigitsOf(int n) {  
    int result = 0;  
    while (n > 0) {  
        result += (n % 10);  
        n /= 10;  
    }  
    return result;  
}
```

137

int n

7

int result

# Functions in Action

```
int sumOfDigitsOf(int n) {  
    int result = 0;  
    while (n > 0) {  
        result += (n % 10);  
        n /= 10;  
    }  
    return result;  
}
```

137

int n

7

int result

# Functions in Action

```
int sumOfDigitsOf(int n) {  
    int result = 0;  
    while (n > 0) {  
        result += (n % 10);  
        n /= 10;  
    }  
    return result;  
}
```

13

int n

7

int result

# Functions in Action

```
int sumOfDigitsOf(int n) {  
    int result = 0;  
    while (n > 0) {  
        result += (n % 10);  
        n /= 10;  
    }  
    return result;  
}
```

13

int n

7

int result

# Functions in Action

```
int sumOfDigitsOf(int n) {  
    int result = 0;  
    while (n > 0) {  
        result += (n % 10);  
        n /= 10;  
    }  
    return result;  
}
```

13

int n

7

int result

# Functions in Action

```
int sumOfDigitsOf(int n) {  
    int result = 0;  
    while (n > 0) {  
        result += (n % 10);  
        n /= 10;  
    }  
    return result;  
}
```

13

int n

10

int result

# Functions in Action

```
int sumOfDigitsOf(int n) {  
    int result = 0;  
    while (n > 0) {  
        result += (n % 10);  
        n /= 10;  
    }  
    return result;  
}
```

13

int n

10

int result



# Functions in Action

```
int sumOfDigitsOf(int n) {  
    int result = 0;  
    while (n > 0) {  
        result += (n % 10);  
        n /= 10;  
    }  
    return result;  
}
```

1

int n

10

int result

# Functions in Action

```
int sumOfDigitsOf(int n) {  
    int result = 0;  
    while (n > 0) {  
        result += (n % 10);  
        n /= 10;  
    }  
    return result;  
}
```

1

int n

10

int result

# Functions in Action

```
int sumOfDigitsOf(int n) {  
    int result = 0;  
    while (n > 0) {  
        result += (n % 10);  
        n /= 10;  
    }  
    return result;  
}
```

1

int n

10

int result

# Functions in Action

```
int sumOfDigitsOf(int n) {  
    int result = 0;  
    while (n > 0) {  
        result += (n % 10);  
        n /= 10;  
    }  
    return result;  
}
```

1

int n

11

int result

# Functions in Action

```
int sumOfDigitsOf(int n) {  
    int result = 0;  
    while (n > 0) {  
        result += (n % 10);  
        n /= 10;  
    }  
    return result;  
}
```

1

int n

11

int result

# Functions in Action

```
int sumOfDigitsOf(int n) {  
    int result = 0;  
    while (n > 0) {  
        result += (n % 10);  
        n /= 10;  
    }  
    return result;  
}
```

0

int n

11

int result

# Functions in Action

```
int sumOfDigitsOf(int n) {  
    int result = 0;  
    while (n > 0) {  
        result += (n % 10);  
        n /= 10;  
    }  
    return result;  
}
```

0

int n

11

int result

# Functions in Action

```
int sumOfDigitsOf(int n) {  
    int result = 0;  
    while (n > 0) {  
        result += (n % 10);  
        n /= 10;  
    }  
    return result;  
}
```

0

int n

11

int result



# Functions in Action

```
int main() {  
    int n = getInteger("Enter an integer: ");  
    int digitSum = sumOfDigitsOf(n);  
    cout << n << " sums to " << digitSum << endl;  
  
    return 0;  
}
```

137

11

# Functions in Action

```
int main() {
    int n = getInteger("Enter an integer: ");
    int digitSum = sumOfDigitsOf(n);
    cout << n << " sums to " << digitSum << endl;

    return 0;
}
```

**137**

**11**

**11**

int digitSum

# Functions in Action

```
int main() {  
    int n = getInteger("Enter an integer: ");  
    int digitSum = sumOfDigitsOf(n);  
    cout << n << " sums to " << digitSum << endl;  
  
    return 0;  
}
```

**137** ← `int n`

**11**  
`int digitSum`

Note that the value of `n` in `main` is unchanged, because `sumOfDigitsOf` got its own copy of `n` that only coincidentally has the same name as the copy in `main`.

# Functions in Action

```
int main() {  
    int n = getInteger("Enter an integer: ");  
    int digitSum = sumOfDigitsOf(n);  
    cout << n << " sums to " << digitSum << endl;  
  
    return 0;  
}
```

137

11

int digitSum

**Time-Out for Announcements!**

# Section Signups

- Section signups go live tomorrow at 5:00PM and are open until Sunday at 5:00PM.
- Sign up using this link:  
**<http://cs198.stanford.edu/section>**
- You need to sign up here even if you're already enrolled on *Axess*; *we don't use Axess for sections in this class.*

# Qt Creator Help Session

- Having trouble getting Qt Creator set up? Katherine is holding a help session on

***Thursday, 8:00PM - 10:00PM***

***Tresidder, First Floor***

- A request from the folks running this session: before showing up, try using the troubleshooting guide and make sure you followed the directions precisely. It's easy to get this wrong, but easy to correct once you identify where you went off-script.

Back to CS106B!



# Thinking Recursively

# Factorials

- The number ***n factorial***, denoted ***n!***, is

$$n \times (n - 1) \times \dots \times 3 \times 2 \times 1$$

- For example:
  - $3! = 3 \times 2 \times 1 = 6.$
  - $4! = 4 \times 3 \times 2 \times 1 = 24.$
  - $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120.$
  - $0! = 1.$  (by definition)
- Factorials show up in unexpected places. We'll see one later this quarter when we talk about sorting algorithms.
- Let's implement a function to compute factorials!

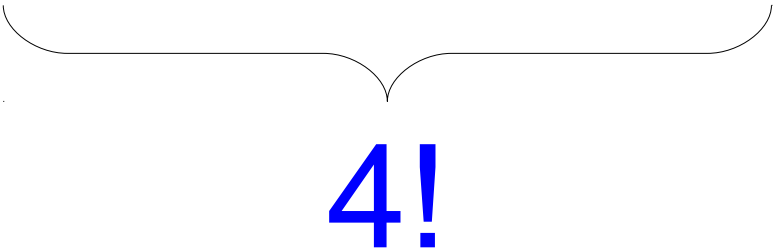
# Computing Factorials

$$5! = 5 \times 4 \times 3 \times 2 \times 1$$

# Computing Factorials

$$5! = 5 \times 4 \times 3 \times 2 \times 1$$

# Computing Factorials

$$5! = 5 \times 4 \times 3 \times 2 \times 1$$


The diagram illustrates the recursive nature of factorials. The expression  $5! = 5 \times 4 \times 3 \times 2 \times 1$  is shown. A curly brace is drawn under the terms  $4 \times 3 \times 2 \times 1$ , and the label  $4!$  is placed directly below the brace, indicating that the product of these four terms is  $4!$ .

# Computing Factorials

$$5! = 5 \times 4!$$

# Computing Factorials

$$5! = 5 \times 4!$$

# Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3 \times 2 \times 1$$



# Computing Factorials

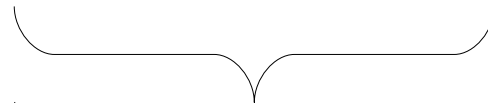
$$5! = 5 \times 4!$$

$$4! = 4 \times 3 \times 2 \times 1$$

# Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3 \times 2 \times 1$$



$$3!$$

# Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

# Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

# Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2 \times 1$$

# Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

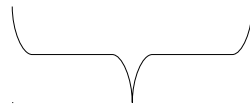
$$3! = 3 \times 2 \times 1$$

# Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2 \times 1$$



$$2!$$

# Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$



# Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

# Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

$$2! = 2 \times 1!$$

# Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

$$2! = 2 \times 1!$$

$$1! = 1 \times 0!$$

# Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

$$2! = 2 \times 1!$$

$$1! = 1 \times 0!$$

$$0! = 1$$

# Another View of Factorials

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n - 1)! & \text{otherwise} \end{cases}$$

# Another View of Factorials

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n - 1)! & \text{otherwise} \end{cases}$$

```
int factorial(int n) {  
    if (n == 0) {  
        return 1;  
    } else {  
        return n * factorial(n - 1);  
    }  
}
```

# Another View of Factorials

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n - 1)! & \text{otherwise} \end{cases}$$

```
int factorial(int n) {  
    if (n == 0) {  
        return 1;  
    } else {  
        return n * factorial(n - 1);  
    }  
}
```

# Recursion in Action

```
int main() {  
    int n = factorial(5);  
    cout << "5! = " << n << endl;  
  
    return 0;  
}
```



# Recursion in Action

```
int main() {  
    int n = factorial(5);  
    cout << "5! = " << n << endl;  
  
    return 0;  
}
```

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        if (n == 0) {
```

```
            return 1;
```

```
        } else {
```

```
            return n * factorial(n - 1);
```

```
        }
```

```
    }
```

5

int n

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        if (n == 0) {
```

```
            return 1;
```

```
        } else {
```

```
            return n * factorial(n - 1);
```

```
        }
```

```
    }
```

5

int n

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        if (n == 0) {
```

```
            return 1;
```

```
        } else {
```

```
            return n * factorial(n - 1);
```

```
        }
```

```
    }
```

5

int n

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        if (n == 0) {
```

```
            return 1;
```

```
        } else {
```

```
            return n * factorial(n - 1);
```

```
        }
```

```
    }
```

5

int n

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        if (n == 0) {
```

```
            return 1;
```

```
        } else {
```

```
            return n * factorial(n - 1);
```

```
        }
```

```
    }
```

5

int n

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        if (n == 0) {
```

```
            return 1;
```

```
        } else {
```

```
            return n * factorial(n - 1);
```

```
        }
```

```
    }
```

5

int n

5

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        if (n == 0) {
```

```
            return 1;
```

```
        } else {
```

```
            return n * factorial(n - 1);
```

```
        }
```

```
    }
```

5

int n

5



# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

```
}
```

```
}
```

4

int n

Every time we call factorial(), we get a new copy of the local variable n that's independent of all the previous copies.

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        int factorial(int n) {
```

```
            if (n == 0) {
```

```
                return 1;
```

```
            } else {
```

```
                return n * factorial(n - 1);
```

```
            }
```

```
        }
```

4

int n

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        int factorial(int n) {
```

```
            if (n == 0) {
```

```
                return 1;
```

```
            } else {
```

```
                return n * factorial(n - 1);
```

```
            }
```

```
        }
```

4

int n

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        int factorial(int n) {
```

```
            if (n == 0) {
```

```
                return 1;
```

```
            } else {
```

```
                return n * factorial(n - 1);
```

```
            }
```

```
        }
```

4

int n

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        int factorial(int n) {
```

```
            if (n == 0) {
```

```
                return 1;
```

```
            } else {
```

```
                return n * factorial(n - 1);
```

```
            }
```

```
        }
```

4

int n

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

4

int n

4

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        int factorial(int n) {
```

```
            if (n == 0) {
```

```
                return 1;
```

```
            } else {
```

```
                return n * factorial(n - 1);
```

```
            }
```

```
        }
```

4

int n

4

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        int factorial(int n) {
```

```
            int factorial(int n) {
```

```
                if (n == 0) {
```

```
                    return 1;
```

```
                } else {
```

```
                    return n * factorial(n - 1);
```

```
                }
```

```
            }
```

3

int n



# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

```
}
```

```
}
```

3

int n

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
    if (n == 0) {
```

```
        return 1;
```

```
    } else {
```

```
        return n * factorial(n - 1);
```

```
    }
```

```
}
```

3

int n

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        int factorial(int n) {
```

```
            int factorial(int n) {
```

```
                if (n == 0) {
```

```
                    return 1;
```

```
                } else {
```

```
                    return n * factorial(n - 1);
```

```
                }
```

```
            }
```

3

int n

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        int factorial(int n) {
```

```
            int factorial(int n) {
```

```
                if (n == 0) {
```

```
                    return 1;
```

```
                } else {
```

```
                    return n * factorial(n - 1);
```

```
                }
```

```
            }
```

3

int n

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
    if (n == 0) {
```

```
        return 1;
```

```
    } else {
```

```
        return n * factorial(n - 1);
```

```
    }
```

```
}
```

3

int n

3

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

```
3
```

```
3
```

```
int n
```

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        int factorial(int n) {
```

```
            int factorial(int n) {
```

```
                int factorial(int n) {
```

```
                    if (n == 0) {
```

```
                        return 1;
```

```
                    } else {
```

```
                        return n * factorial(n - 1);
```

```
                    }
```

```
                }
```

2

int n

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

```
}
```

```
}
```

2

int n



# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

```
}
```

```
}
```

2

int n

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
    if (n == 0) {
```

```
        return 1;
```

```
    } else {
```

```
        return n * factorial(n - 1);
```

```
    }
```

```
}
```

2

int n

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        int factorial(int n) {
```

```
            int factorial(int n) {
```

```
                int factorial(int n) {
```

```
                    if (n == 0) {
```

```
                        return 1;
```

```
                    } else {
```

```
                        return n * factorial(n - 1);
```

```
                    }
```

```
                }
```

2

int n

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        int factorial(int n) {
```

```
            int factorial(int n) {
```

```
                int factorial(int n) {
```

```
                    if (n == 0) {
```

```
                        return 1;
```

```
                    } else {
```

```
                        return n * factorial(n - 1);
```

```
                    }
```

```
                }
```

2

int n

2

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
    if (n == 0) {
```

```
        return 1;
```

```
    } else {
```

```
        return n * factorial(n - 1);
```

```
    }
```

2

int n

2

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
    if (n == 0) {
```

```
        return 1;
```

```
    } else {
```

```
        return n * factorial(n - 1);
```

```
    }
```

```
}
```

1

int n

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

```
}
```

```
}
```

1

int n

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
    if (n == 0) {
```

```
        return 1;
```

```
    } else {
```

```
        return n * factorial(n - 1);
```

```
    }
```

```
}
```

1

int n



# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

```
}
```

```
}
```

1

int n

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

```
}
```

```
}
```

1

int n

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

```
}
```

```
}
```

1

int n

1

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
    if (n == 0) {
```

```
        return 1;
```

```
    } else {
```

```
        return n * factorial(n - 1);
```

```
    }
```

```
}
```

1

int n

1

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        int factorial(int n) {
```

```
            int factorial(int n) {
```

```
                int factorial(int n) {
```

```
                    int factorial(int n) {
```

```
                        int factorial(int n) {
```

```
                            if (n == 0) {
```

```
                                return 1;
```

```
                            } else {
```

```
                                return n * factorial(n - 1);
```

```
                            }
```

```
                        }
```

0

int n

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        int factorial(int n) {
```

```
            int factorial(int n) {
```

```
                int factorial(int n) {
```

```
                    int factorial(int n) {
```

```
                        int factorial(int n) {
```

```
                            if (n == 0) {
```

```
                                return 1;
```

```
                            } else {
```

```
                                return n * factorial(n - 1);
```

```
                            }
```

```
                        }
```

```
0
```

```
int n
```

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

```
}
```

```
}
```

0

int n

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
    if (n == 0) {
```

```
        return 1;
```

```
    } else {
```

```
        return n * factorial(n - 1);
```

```
    }
```

```
}
```

1

int n

1



# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

```
}
```

1

1

1

int n

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

```
}
```

1

1

1

int n

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

```
}
```

**1** × **1**

**1**

int n

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

```
}
```

**1**

**1**

int n

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        int factorial(int n) {
```

```
            int factorial(int n) {
```

```
                int factorial(int n) {
```

```
                    if (n == 0) {
```

```
                        return 1;
```

```
                    } else {
```

```
                        return n * factorial(n - 1);
```

```
                    }
```

```
                }
```

2

int n

2

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

```
}
```

2

1

2

int n

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

**2**

**1**

**2**

int n

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
    if (n == 0) {
```

```
        return 1;
```

```
    } else {
```

```
        return n * factorial(n - 1);
```

**2** × **1**

**2**

int n



# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        int factorial(int n) {
```

```
            int factorial(int n) {
```

```
                int factorial(int n) {
```

```
                    if (n == 0) {
```

```
                        return 1;
```

```
                    } else {
```

```
                        return n * factorial(n - 1);
```

```
                    }
```

```
                }
```

2

int n

2

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        int factorial(int n) {
```

```
            int factorial(int n) {
```

```
                if (n == 0) {
```

```
                    return 1;
```

```
                } else {
```

```
                    return n * factorial(n - 1);
```

```
                }
```

```
            }
```

3

int n

3

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

3

2

3

int n

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        int factorial(int n) {
```

```
            int factorial(int n) {
```

```
                if (n == 0) {
```

```
                    return 1;
```

```
                } else {
```

```
                    return n * factorial(n - 1);
```

**3**

**2**

**3**

int n

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

**3** × **2**

**3**

int n

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        int factorial(int n) {
```

```
            int factorial(int n) {
```

```
                if (n == 0) {
```

```
                    return 1;
```

```
                } else {
```

```
                    return n * factorial(n - 1);
```

```
                }
```

```
            }
```

3

int n

6

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

4

int n

4

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

4

int n

4

6



# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

4

int n

4

6

# Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
if (n == 0) {
```

```
return 1;
```

```
} else {
```

```
return n * factorial(n - 1);
```

4

int n

4

×

6

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        int factorial(int n) {
```

```
            if (n == 0) {
```

```
                return 1;
```

```
            } else {
```

```
                return n * factorial(n - 1);
```

4

int n

24

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        if (n == 0) {
```

```
            return 1;
```

```
        } else {
```

```
            return n * factorial(n - 1);
```

```
        }
```

```
    }
```

5

int n

5

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        if (n == 0) {
```

```
            return 1;
```

```
        } else {
```

```
            return n * factorial(n - 1);
```

```
        }
```

```
    }
```

5

int n

5

24

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        if (n == 0) {
```

```
            return 1;
```

```
        } else {
```

```
            return n * factorial(n - 1);
```

```
        }
```

```
    }
```

5

int n

5

24

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        if (n == 0) {
```

```
            return 1;
```

```
        } else {
```

```
            return n * factorial(n - 1);
```

```
        }
```

```
    }
```

5

int n

5

×

24

# Recursion in Action

```
int main() {
```

```
    int factorial(int n) {
```

```
        if (n == 0) {
```

```
            return 1;
```

```
        } else {
```

```
            return n * factorial(n - 1);
```

```
        }
```

```
    }
```

5

int n

120



# Recursion in Action

```
int main() {  
    int n = factorial(5);  
    cout << "5! = " << n << endl;  
  
    return 0;  
}
```

# Recursion in Action

```
int main() {  
    int n = factorial(5);  
    cout << "5! = " << n << endl;  
  
    return 0;  
}
```

120

int n

# Thinking Recursively

- Solving a problem with recursion requires two steps.
- First, determine how to solve the problem for simple cases.
  - This is called the ***base case***.
- Second, determine how to break down larger cases into smaller instances.
  - This is called the ***recursive step***.

# Recap from Today

- The C++ compiler reads from the top of the program to the bottom. You cannot call a function that hasn't either been prototyped or defined before the call site.
- Each time you call a function, C++ gives you a fresh copy of all the local variables in that function. Those variables are independent of any other variables with the same name found elsewhere.
- You can split a number into “everything but the last digit” and “the last digit” by dividing and modding by 10.
- A ***recursive function*** is one that calls itself. It has a ***base case*** to handle easy cases and a ***recursive step*** to turn bigger versions of the problem into smaller ones.
- Functions can be written both iteratively and recursively.

# Your Action Items

- ***Read Chapter 1 and Chapter 2.***
  - We're still easing into C++. These chapters talk about the basics and the mechanics of function call and return.
- ***Read Chapter 7.***
  - We've just started talking about recursion. There's tons of goodies in that chapter.
- ***Sign up for section.***
  - The link goes out tomorrow afternoon.
- ***Work on Assignment 0.***
  - Just over a third of you are already done! Exciting!

# Next Time

- ***Strings and Streams***
  - Representing and Manipulating Text.
  - Recursion on Text.
  - File I/O in C++.
- ***More Recursion***
  - Getting more comfortable with this strategy.