

# Functions in C++

# Outline for Today

- ***Functions in C++***
  - How C++ organizes code.
- ***Forward Declarations***
  - A charming, endearing quirk of C++.
- ***Some Simple Functions***
  - Getting comfortable with the language.
- ***Intro to Recursion***
  - A new perspective on problem-solving.

# Functions in C++

# C++ Functions

- Functions in C++ are similar to methods in Java and functions in JavaScript / Python:
  - They're pieces of code that perform tasks.
  - They (optionally) take parameters.
  - They (optionally) return a value.
- Here's some C++ functions:

```
double areaOfCircle(double r) {  
    return M_PI * r * r;  
}
```

Coming from JavaScript or Python? Note that you need to assign types to variables, parameters, and return values.

```
void printBiggerOf(int a, int b) {  
    if (a > b) {  
        cout << a << endl;  
    } else {  
        cout << b << endl;  
    }  
}
```

# The main Function

- A C++ program begins execution in a function called `main` with the following signature:

```
int main() {  
    /* ... code to execute ... */  
    return 0;  
}
```

- By convention, `main` should return 0 unless the program encounters an error.

The function `main` returns an integer.  
Curious where that integer goes?  
Come talk to me after class!

# 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?



*Dramatic Reenactment*

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.

What other functions can we write?

# Factorials

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

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

- For example:
  - $3! = 3 \times 2 \times 1 = 6$ .
  - $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$
  - $0! = 1$  (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!

# 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).*

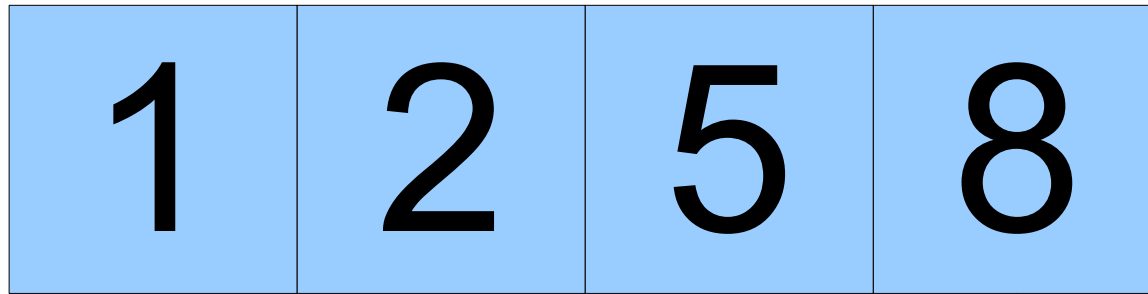
- Let's write a function

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

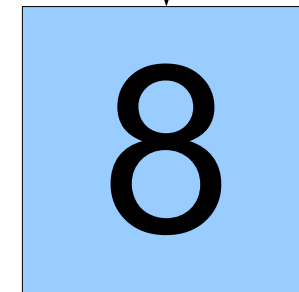
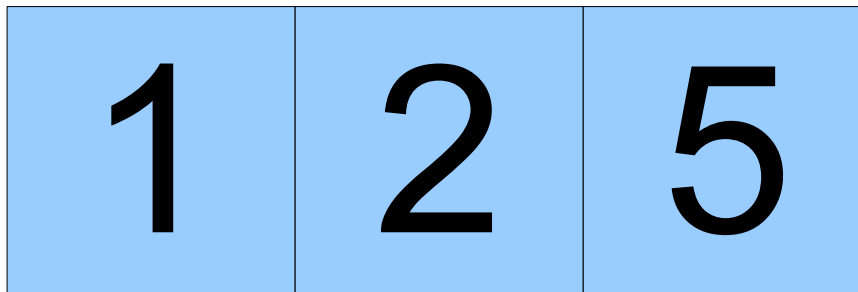
that takes in a number and returns the sum of its digits.



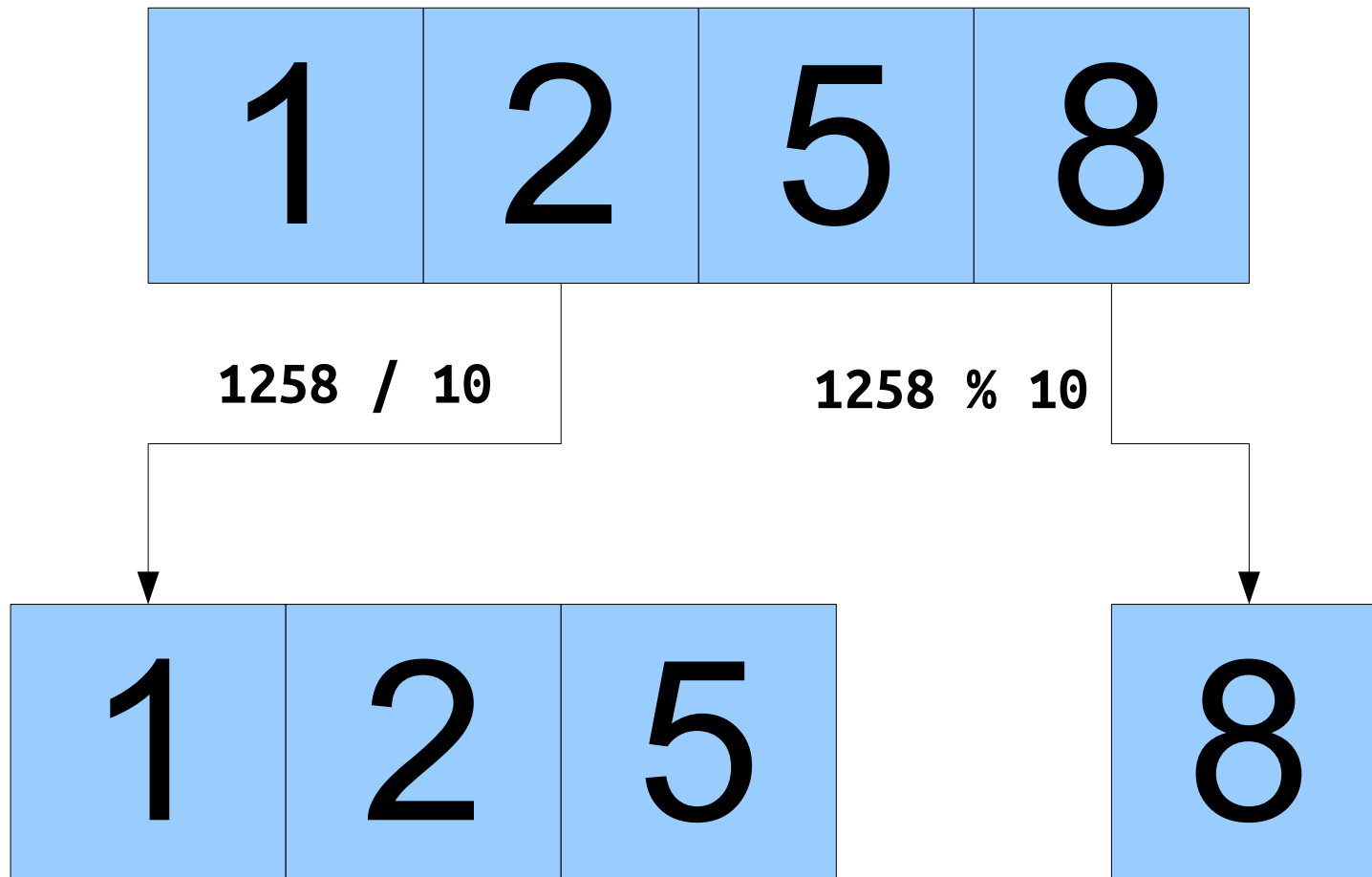
# Working One Digit at a Time



**1258 % 10**



# Working One Digit at a Time



# Digital Roots

- The ***digital root*** is the number you get by repeatedly summing the digits of a number until you're down to a single digit.
- The digital root of 5 is 5.
- The digital root of 25 is found as follows:

$$2 + 5 = 7$$

So the digital root of 25 is 7.

- The digital root of 137 is found as follows:

$$1 + 3 + 7 = 11$$

$$1 + 1 = 2$$

- So the digital root of 137 is 2.

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? We're holding a help session on

***Wednesday, 7PM - 9PM***  
***Tresidder, First Floor***

- A note: the printed version of the Assignment 0 handout mentioned that this would be held on Thursday. That's an error; the session is tonight.
- 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.

# CS 100B

- ***CS 100B*** is a new, optional, one-unit add-on to CS106B that is designed to provide extra practice with the material.
- It's “in addition to” rather than “instead of” your discussion section.
- Interested? Apply using [\*\*this link\*\*](#).

Back to CS106B!



# Thinking Recursively

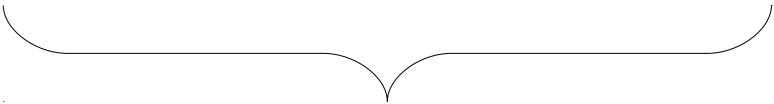
# Factorial Revisited

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

# Factorial Revisited

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

# Factorial Revisited

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

$$4!$$

# Factorial Revisited

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

# Factorial Revisited

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

# Factorial Revisited

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

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

# Factorial Revisited

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

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



# Factorial Revisited

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

$$4! = 4 \times \underbrace{3 \times 2 \times 1}_{3!}$$

# Factorial Revisited

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

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

# Factorial Revisited

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

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

# Factorial Revisited

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

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

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

# Factorial Revisited

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

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

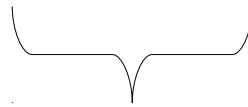
$$3! = 3 \times 2 \times 1$$

# Factorial Revisited

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

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

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



$2!$

# Factorial Revisited

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

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

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

# Factorial Revisited

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

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

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



# Factorial Revisited

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

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

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

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

# Factorial Revisited

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

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

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

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

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

# Factorial Revisited

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

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

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

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

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

$$0! = 1$$

# 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



# 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

int n

3

# 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

int n

1

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

×

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) {
```

```
                    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

1

# 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

1

# 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

×

1

# 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

int n

3

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

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

×

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

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***.

# Summing Up Digits

- One way to compute the sum of the digits of a number is shown here:

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

- How would we rewrite this function recursively?

# Summing Up Digits

1	2	5	8
---	---	---	---

The sum of the digits of  
this number is equal to...

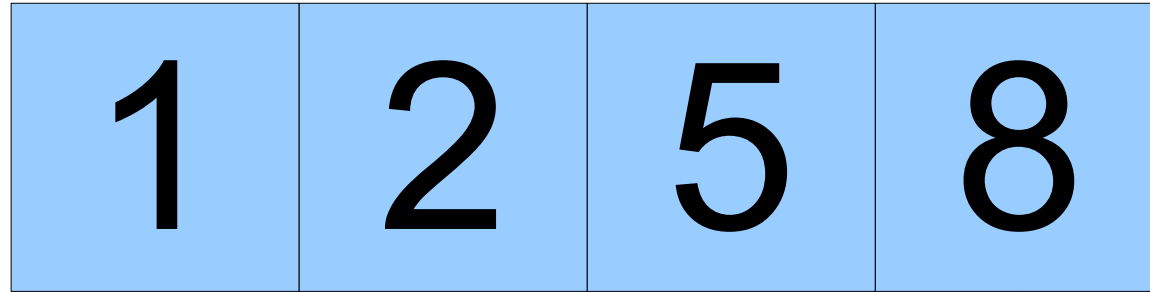
the sum of the digits of  
this number...

1	2	5
---	---	---

plus this number.

8
---

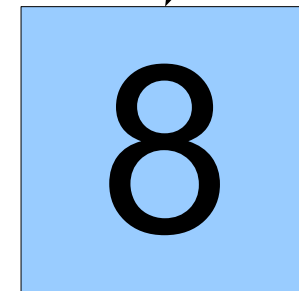
# Summing Up Digits



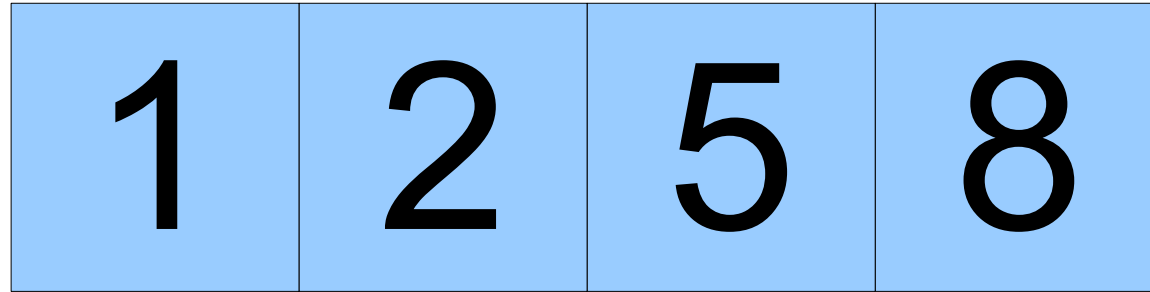
sumofDigitsof(n)  
is equal to...

the sum of the digits of  
this number...

plus this number.



# Summing Up Digits



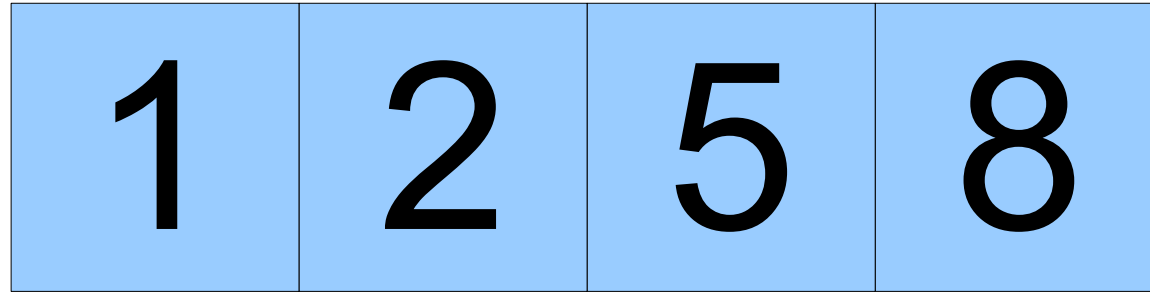
sumofDigitsof(n)  
is equal to...

sumofDigitsof(n / 10)

plus this number.



# Summing Up Digits



sumofDigitsof(n)  
is equal to...

sumofDigitsof( $n / 10$ )

+ ( $n \% 10$ )



# Summing Up Digits

- Here's our recursive rewrite:

```
int sumOfDigitsOf(int n) {  
    if (n < 10) {  
        return n;  
    } else {  
        return sumOfDigitsOf(n / 10) + (n % 10);  
    }  
}
```

- Notice the structure:
  - If the problem is sufficiently simple, solve it directly.
  - If not, reduce it to a smaller instance and solve it.



# Tracing the Recursion

```
int main() {  
    int sum = sumOfDigitsOf(137);  
    cout << "Sum is " << sum << endl;  
}
```

# Tracing the Recursion

```
int main() {  
    int sum = sumOfDigitsOf(137);  
    cout << "Sum is " << sum << endl;  
}
```

# Tracing the Recursion

```
int main() {
```

```
    int sumOfDigitsOf(int n) {  
        if (n < 10) {  
            return n;  
        } else {  
            return sumOfDigitsOf(n / 10) + (n % 10);  
        }  
    }
```

int n    137

# Tracing the Recursion

```
int main() {
```

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

```
        if (n < 10) {
```

```
            return n;
```

```
        } else {
```

```
            return sumOfDigitsOf(n / 10) + (n % 10);
```

```
        }
```

```
    }
```

int n    137

# Tracing the Recursion

```
int main() {
```

```
    int sumOfDigitsOf(int n) {  
        if (n < 10) {  
            return n;  
        } else {  
            return sumOfDigitsOf(n / 10) + (n % 10);  
        }  
    }  
}
```

int n    137

# Tracing the Recursion

```
int main() {
```

```
    int sumOfDigitsOf(int n) {  
        if (n < 10) {  
            return n;  
        } else {  
            return sumOfDigitsOf(n / 10) + (n % 10);  
        }  
    }
```

int n    137

# Tracing the Recursion

```
int main() {
```

```
    int sumOfDigitsOf(int n) {  
        if (n < 10) {  
            return n;  
        } else {  
            return sumOfDigitsOf(n / 10) + (n % 10);  
        }  
    }  
}
```

int n    137

# Tracing the Recursion

```
int main() {
```

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

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

```
            if (n < 10) {
```

```
                return n;
```

```
            } else {
```

```
                return sumOfDigitsOf(n / 10) + (n % 10);
```

```
            }
```

```
        }
```

int n 13



# Tracing the Recursion

```
int main() {
```

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

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

```
            if (n < 10) {
```

```
                return n;
```

```
            } else {
```

```
                return sumOfDigitsOf(n / 10) + (n % 10);
```

```
            }
```

```
        }
```

```
int n 13
```

# Tracing the Recursion

```
int main() {
```

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

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

```
            if (n < 10) {
```

```
                return n;
```

```
            } else {
```

```
                return sumOfDigitsOf(n / 10) + (n % 10);
```

```
            }
```

```
        }
```

```
int n 13
```

# Tracing the Recursion

```
int main() {
```

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

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

```
            if (n < 10) {
```

```
                return n;
```

```
            } else {
```

```
                return sumOfDigitsOf(n / 10) + (n % 10);
```

int n 13

# Tracing the Recursion

```
int main() {
```

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

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

```
            if (n < 10) {
```

```
                return n;
```

```
            } else {
```

```
                return sumOfDigitsOf(n / 10) + (n % 10);
```

```
            }
```

```
        }
```

int n 13

# Tracing the Recursion

```
int main() {
```

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

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

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

```
                if (n < 10) {
```

```
                    return n;
```

```
                } else {
```

```
                    return sumOfDigitsOf(n / 10) + (n % 10);
```

```
                }
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

int n

1

107

10

# Tracing the Recursion

```
int main() {
```

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

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

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

```
                if (n < 10) {
```

```
                    return n;
```

```
                } else {
```

```
                    return sumOfDigitsOf(n / 10) + (n % 10);
```

```
                }
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

int n

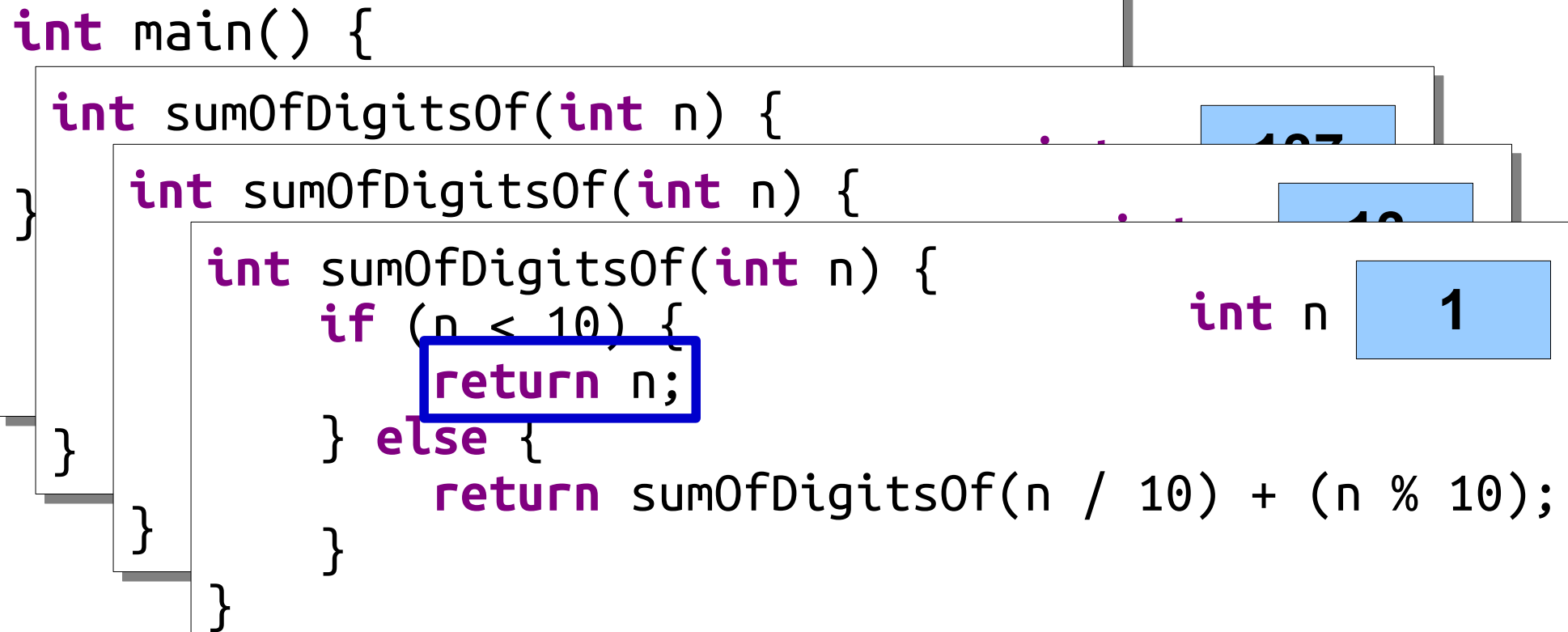
1

107

10

# Tracing the Recursion

```
int main() {  
    int sumOfDigitsOf(int n) {  
        int sumOfDigitsOf(int n) {  
            int sumOfDigitsOf(int n) {  
                if (n < 10) {  
                    return n;  
                } else {  
                    return sumOfDigitsOf(n / 10) + (n % 10);  
                }  
            }  
        }  
    }  
}
```



The diagram illustrates the recursive calls for the function `sumOfDigitsOf` with the input `123`. The stack of frames shows the following values of `n` in the function calls:

- Top frame: `n = 123`
- Middle frame: `n = 12`
- Bottom frame: `n = 1`

The current frame (for `n = 1`) is highlighted, and the `return n;` statement is enclosed in a blue box, indicating the base case of the recursion.

# Tracing the Recursion

```
int main() {
```

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

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

```
            if (n < 10) {
```

```
                return n;
```

```
            } else {
```

```
                return sumOfDigitsOf(n / 10) + (n % 10);
```

int n 13

1



# Tracing the Recursion

```
int main() {
```

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

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

```
            if (n < 10) {
```

```
                return n;
```

```
            } else {
```

```
                return sumOfDigitsOf(n / 10) + (n % 10);
```

1

int n 13

# Tracing the Recursion

```
int main() {
```

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

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

```
            if (n < 10) {
```

```
                return n;
```

```
            } else {
```

```
                return sumOfDigitsOf(n / 10) + (n % 10);
```

1

+

3

int n

13

# Tracing the Recursion

```
int main() {
```

```
    int sumOfDigitsOf(int n) {  
        if (n < 10) {  
            return n;  
        } else {  
            return sumOfDigitsOf(n / 10) + (n % 10);  
        }  
    }  
}
```

int n

137

4

# Tracing the Recursion

```
int main() {
```

```
    int sumOfDigitsOf(int n) {  
        if (n < 10) {  
            return n;  
        } else {  
            return sumOfDigitsOf(n / 10) + (n % 10);  
        }  
    }
```

int n 137

4

# Tracing the Recursion

```
int main() {
```

```
    int sumOfDigitsOf(int n) {  
        if (n < 10) {  
            return n;  
        } else {  
            return sumOfDigitsOf(n / 10) + (n % 10);  
        }  
    }
```

int n

137

4

+

7

# Tracing the Recursion

```
int main() {  
    int sum = sumOfDigitsOf(137);  
    cout << "Sum is " << sum << endl;  
}
```

**11**

# Recap from Today

- In C++, to call a function, that function must either be defined above the call site or **forward-declared** before the call site.
- Execution in C++ programs begins in a function called `main`.
- 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 of the textbook for more background on C++ basics and writing functions.
- Read Chapter 7 of the textbook for more exposition on recursion.
- Keep an eye out for the section signup link, which will get sent out tomorrow afternoon.
- Aim to complete Assignment 0 by Friday.
  - 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++.