

Getting Started in C++

Outline for Today

- ***Functions in C++***
 - How C++ organizes code, and some Endearing C++ Quirks.
- ***Writing Functions***
 - Getting comfortable with the language.
- ***Strings in C++***
 - Working with text.

Functions in C++

```

/*          C++ Version          */
double areaOfCircle(double r) {
    return M_PI * r * r;
}

int maxOf(int first, int second) {
    if (first > second) {
        return first;
    }
    return second;
}

void printNumber(int n) {
    cout << "I like " << n << endl;
}

```

```

"""          Python Version          """
def areaOfCircle(r):
    return math.pi * r * r

def maxOf(first, second):
    if first > second:
        return first
    return second

def printNumber(n):
    print("I like " + str(n))

```

```

/*          Java Version          */
private double areaOfCircle(double r) {
    return Math.PI * r * r;
}

private int maxOf(int first, int second) {
    if (first > second) {
        return first;
    }
    return second;
}

private void printNumber(int n) {
    System.out.println("I like " + n);
}

```

```

//          JavaScript Version
function areaOfCircle(r) {
    return Math.PI * r * r;
}

function maxOf(first, second) {
    if (first > second) {
        return first;
    }
    return second;
}

function printNumber(n) {
    console.log("I like " + n);
}

```

```

/*          C++ Version          */
double areaOfCircle(double r) {
    return M_PI * r * r;
}

int maxOf(int first, int second) {
    if (first > second) {
        return first;
    }
    return second;
}

void printNumber(int n) {
    cout << "I like " << n << endl;
}

```

```

"""          Python Version          """
def areaOfCircle(r):
    return math.pi * r * r

def maxOf(first, second):
    if first > second:
        return first
    return second

def printNumber(n):
    print("I like " + str(n))

```

Functions in C++ work like functions in Python/JavaScript or like methods in Java. They (optionally) take in parameters, perform a calculation, then (optionally) return a value.

```

/*          JavaScript Version          */
private double areaOfCircle(r) {
    return Math.PI * r * r;
}

private int maxOf(first, second) {
    if (first > second) {
        return first;
    }
    return second;
}

private void printNumber(n) {
    System.out.println("I like " + n);
}

```

```

//          JavaScript Version          //
function areaOfCircle(r) {
    return Math.PI * r * r;
}

function maxOf(first, second) {
    if (first > second) {
        return first;
    }
    return second;
}

function printNumber(n) {
    console.log("I like " + n);
}

```

```

/*          C++ Version          */
double areaOfCircle(double r) {
    return M_PI * r * r;
}

int maxOf(int first, int second) {
    if (first > second) {
        return first;
    }
    return second;
}

void printNumber(int n) {
    cout << "I like " << n << endl;
}

```

```

"""          Python Version          """
def areaOfCircle(r):
    return math.pi * r * r

def maxOf(first, second):
    if first > second:
        return first
    return second

def printNumber(n):
    print("I like " + str(n))

```

All variables in C++ need a type. Some common types include **int** (integer), **double** (real number), and **bool** (true/false),

```

/*          Java Version          */
private double areaOfCircle(double r) {
    return Math.PI * r * r;
}

private int maxOf(int first, int second) {
    if (first > second) {
        return first;
    }
    return second;
}

private void printNumber(int n) {
    System.out.println("I like " + n);
}

```

```

//          JavaScript Version          //
function areaOfCircle(r) {
    return Math.PI * r * r;
}

function maxOf(first, second) {
    if (first > second) {
        return first;
    }
    return second;
}

function printNumber(n) {
    console.log("I like " + n);
}

```

```

/*          C++ Version          */
double areaOfCircle(double r) {
    return M_PI * r * r;
}

int maxOf(int first, int second) {
    if (first > second) {
        return first;
    }
    return second;
}

void printNumber(int n) {
    cout << "I like " << n << endl;
}

```

```

"""          Python Version          """
def areaOfCircle(r):
    return math.pi * r * r

def maxOf(first, second):
    if first > second:
        return first
    return second

def printNumber(n):
    print("I like " + str(n))

```

We only mention a variable's type once, when we introduce that variable. Afterwards, we don't repeat that information and just use the name.

```

/*          Java Version          */
private double areaOfCircle(double r) {
    return Math.PI * r * r;
}

private int maxOf(int first, int second) {
    if (first > second) {
        return first;
    }
    return second;
}

private void printNumber(int n) {
    System.out.println("I like " + n);
}

```

```

//          JavaScript Version          //
function areaOfCircle(r) {
    return Math.PI * r * r;
}

function maxOf(first, second) {
    if (first > second) {
        return first;
    }
    return second;
}

function printNumber(n) {
    console.log("I like " + n);
}

```

```

/*          C++ Version          */
double areaOfCircle(double r) {
    return M_PI * r * r;
}

int maxOf(int first, int second) {
    if (first > second) {
        return first;
    }
    return second;
}

void printNumber(int n) {
    cout << "I like " << n << endl;
}

```

```

"""          Python Version          """
def areaOfCircle(r):
    return math.pi * r * r

def maxOf(first, second):
    if first > second:
        return first
    return second

def printNumber(n):
    print("I like " + str(n))

```

```

/*          Java Version          */
private double areaOfCircle(double r) {
    return Math.PI * r * r;
}

private int maxOf(int first, int second) {
    if (first > second) {
        return first;
    }
    return second;
}

private void printNumber(int n) {
    System.out.println("I like " + n);
}

```

```

//          JavaScript Version          //
function areaOfCircle(r) {
    return Math.PI * r * r;
}

function maxOf(first, second) {
    if (first > second) {
        return first;
    }
    return second;
}

function printNumber(n) {
    console.log("I like " + n);
}

```

You define a function by writing

```

return-type fn-name(params) {
    // ... code goes here ...
}

```



```

/*          C++ Version          */
double areaOfCircle(double r) {
    return M_PI * r * r;
}

int maxOf(int first, int second) {
    if (first > second) {
        return first;
    }
    return second;
}

void printNumber(int n) {
    cout << "I like " << n << endl;
}

```

```

"""          Python Version          """
def areaOfCircle(r):
    return math.pi * r * r

def maxOf(first, second):
    if first > second:
        return first
    return second

def printNumber(n):
    print("I like " + str(n))

```

```

/*          Java Version          */
private double areaOfCircle(double r) {
    return Math.PI * r * r;
}

private int maxOf(int first, int second) {
    if (first > second) {
        return first;
    }
    return second;
}

private void printNumber(int n) {
    System.out.println("I like " + n);
}

```

```

//          JavaScript Version          //
function areaOfCircle(r) {
    return Math.PI * r * r;
}

function maxOf(first, second) {
    if (first > second) {
        return first;
    }
    return second;
}

function printNumber(n) {
    console.log("I like " + n);
}

```

If a function does not return a value, its return type should be the cool-but-scary-sounding **void**.

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.
- Curious why `main` returns an `int` and why it should be `0`? Come chat with me after class today!

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 encounter this issue. What should you do when it happens?



Dramatic Reenactment

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

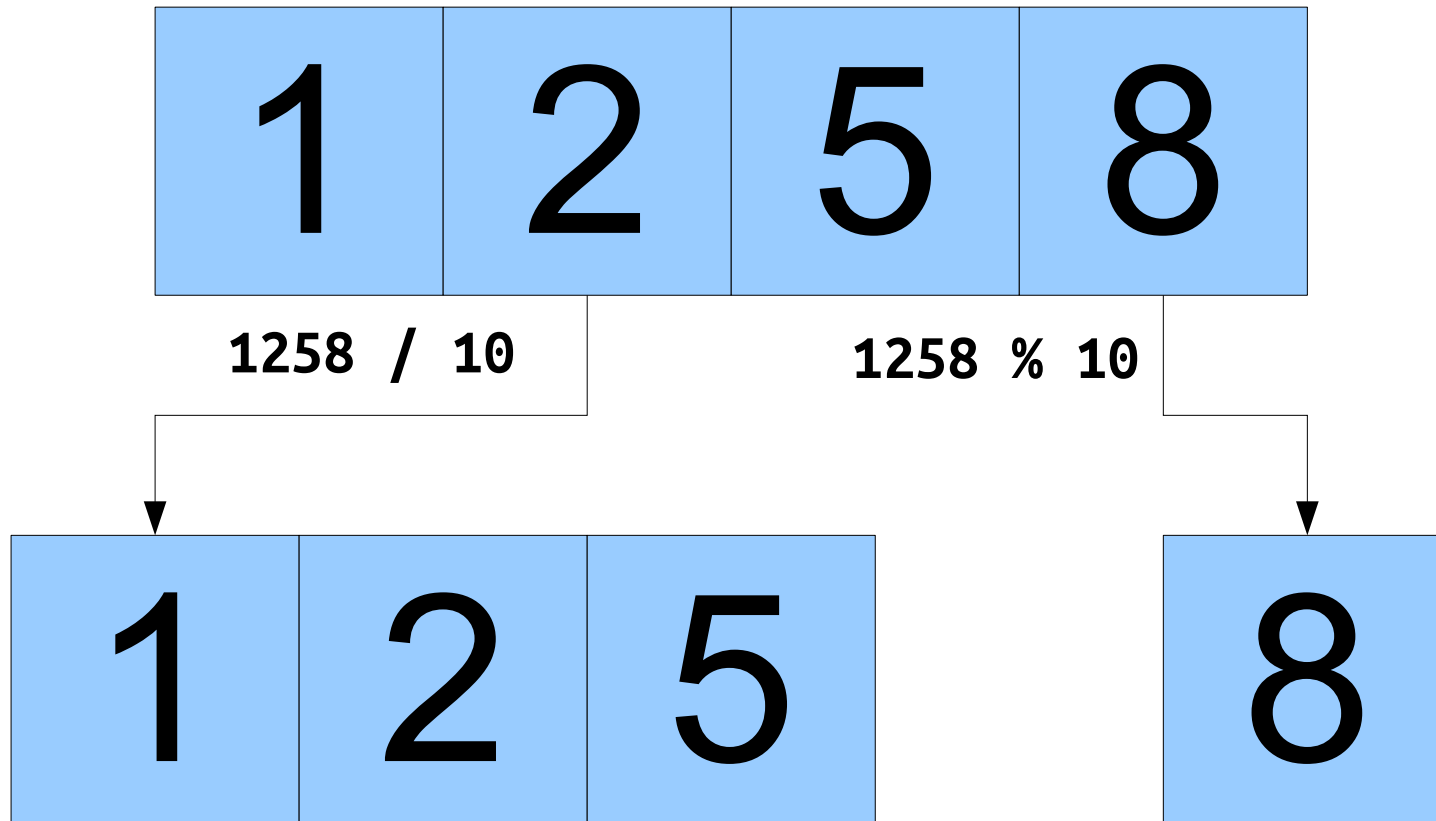
Summing Up Digits

- 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 Chapter 1.7 of 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;  
}
```



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

137

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


Functions in Action

137

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

Functions in Action

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

137

137

int n

Functions in Action

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

137

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

137

int n

Functions in Action

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

137

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

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

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

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

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;  
}
```

127

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;  
}
```

127

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;  
}
```

127

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;  
}
```

127

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;  
}
```

127

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;  
}
```

127

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;  
}
```

127

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;  
}
```

127

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;  
}
```

127

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;  
}
```

127

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;  
}
```

127

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;  
}
```

127

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;  
}
```

127

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;  
}
```

127

0

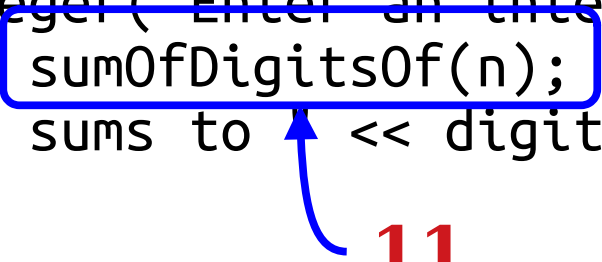
int n

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

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!

The considering_cs List

- The CS department has a mailing list announcing events, programs, and opportunities for folks considering majoring in CS.
- Sign up using [*this link*](#), and please spread the word that this exists! It's a fantastic resource.

Section Signups

- Section signups go live tomorrow at 5:00PM and are open until Sunday at 5:00PM.
- Sign up using this link:
<https://cs198.stanford.edu/cs198/auth/default.aspx>
- 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? Jonathan will be running a Qt Creator help session this evening from 7:00PM - 9:00PM in Room 353 of the Durand building.
- A request: Before showing up, use 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!

Strings in C++

C++ Strings

- To use strings, you need to add the line
`#include <string>`
to the top of your program to import the strings library. You'll get Cruel and Unusual Error Messages if you forget to do this.
- Then, you can do whatever stringy things you want! Here's some examples...

```

/*          C++ Version          */
string s = "Elena Kagan";
s += ", joined " + to_string(2010);

char start = s[0];
char end   = s[s.length() - 1];

if (s.find("e") != string::npos) {
    string first = s.substr(0, 5);
    string last  = s.substr(6);
}

if (s == "Sonia Sotomayor") {
    cout << "John Roberts" << endl;
}

```

```

"""          Python Version          """
s = "Elena Kagan"
s += ", joined " + str(2010)

start = s[0]
end   = s[-1]

if 'e' in s:
    first = s[0:5]
    last  = s[6:]

if s == "Sonia Sotomayor":
    print("John Roberts")

```

```

/*          Java Version          */
String s = "Elena Kagan";
s += ", joined " + 2010;

char start = s.charAt(0);
char end   = s.charAt(s.length() - 1);

if (s.indexOf("e") != -1) {
    String first = s.substring(0, 5);
    String last  = s.substring(6);
}

if (s.equals("Sonia Sotomayor")) {
    System.out.println("John Roberts");
}

```

```

//          JavaScript Version
let s = "Elena Kagan";
s += ", joined " + 2010;

let start = s[0];
let end   = s[s.length - 1];

if (s.indexOf("e") != -1) {
    let first = s.substring(0, 5);
    let last  = s.substring(6);
}

if (s === "Sonia Sotomayor") {
    console.log("John Roberts");
}

```

```
/*          C++ Version          */
string s = "Elena Kagan";
s += ", joined " + to_string(2010);

char start = s[0];
char end    = s[s.length() - 1];

if (s.find("e") != string::npos) {
    string first = s.substr(0, 5);
    string last  = s.substr(6);
}

if (s == "Sonia Sotomayor") {
    cout << "John Roberts" << endl;
}
```

```
"""          Python Version          """
s = "Elena Kagan"
s += ", joined " + str(2010)

start = s[0]
end    = s[-1]

if 'e' in s:
    first = s[0:5]
    last  = s[6:]

if s == "Sonia Sotomayor":
    print("John Roberts")
```

```
/*          Java Version          */
String s = "Elena Kagan";
s += ", joined " + 2010;

char start = s.charAt(0);
char end    = s.charAt(s.length() - 1);

if (s.indexOf("e") != -1) {
    String first = s.substring(0, 5);
    String last  = s.substring(6);
}

if (s.equals("Sonia Sotomayor")) {
    System.out.println("John Roberts");
}
```

```
//          JavaScript Version
let s = "Elena Kagan";
s += ", joined " + 2010;

let start = s[0];
let end    = s[s.length - 1];

if (s.indexOf("e") != -1) {
    let first = s.substring(0, 5);
    let last  = s.substring(6);
}

if (s === "Sonia Sotomayor") {
    console.log("John Roberts");
}
```

```
/*          C++ Version          */
string s = "Elena Kagan";
s += ", joined " + to_string(2010);
char start = s[0];
char end    = s[s.length() - 1];
if (s.find("e") != string::npos) {
    string first = s.substr(0, 5);
    string last  = s.substr(6);
}
if (s == "Sonia Sotomayor") {
    cout << "John Roberts" << endl;
}
```

```
"""          Python Version          """
s = "Elena Kagan"
s += ", joined " + str(2010)
start = s[0]
end    = s[s.length() - 1]
if (s.find("e") != string::npos) {
    string first = s.substr(0, 5);
    string last  = s.substr(6);
}
if (s == "Sonia Sotomayor") {
    cout << "John Roberts" << endl;
}
```

C++ strings must be declared using double quotes rather than single quotes.

```
/*          Java Version          */
String s = "Elena Kagan";
s += ", joined " + 2010;
char start = s.charAt(0);
char end    = s.charAt(s.length() - 1);
if (s.indexOf("e") != -1) {
    String first = s.substring(0, 5);
    String last  = s.substring(6);
}
if (s.equals("Sonia Sotomayor")) {
    System.out.println("John Roberts");
}
```

```
//          JavaScript Version
let s = "Elena Kagan";
s += ", joined " + 2010;
let start = s[0];
let end    = s[s.length - 1];
if (s.indexOf("e") != -1) {
    let first = s.substring(0, 5);
    let last  = s.substring(6);
}
if (s === "Sonia Sotomayor") {
    console.log("John Roberts");
}
```

```
/*          C++ Version          */
string s = "Elena Kagan";
s += ", joined " + to_string(2010);
char start = s[0];
char end   = s[s.length() - 1];
if (s.find("e") != string::npos) {
    string first = s.substr(0, 5);
    string last  = s.substr(6);
}
if (s == "Sonia Sotomayor") {
    cout << "John Roberts" << endl;
}
```

```
/*          Java Version          */
String s = "Elena Kagan";
s += ", joined " + 2010;
char start = s.charAt(0);
char end   = s.charAt(s.length() - 1);
if (s.indexOf("e") != -1) {
    String first = s.substring(0, 5);
    String last  = s.substring(6);
}
if (s.equals("Sonia Sotomayor")) {
    System.out.println("John Roberts");
}
```

```
"""          Python Version          """
s = "Elena Kagan"
s += ", joined " + str(2010)
start = s[0]
end   = s[-1]
if 'e' in s:
```

You can use + and += to append to a string. You can only append other strings or characters. Use the to_string function to convert data to strings.

```
"""          Python Version          """
s = "Elena Kagan"
s += ", joined " + 2010;
let start = s[0];
let end   = s[s.length - 1];
if (s.indexOf("e") != -1) {
    let first = s.substring(0, 5);
    let last  = s.substring(6);
}
if (s === "Sonia Sotomayor") {
    console.log("John Roberts");
}
```

```

/*          C++ Version          */
string s = "Elena Kagan";
s += ", joined " + to_string(2010);
char start = s[0];
char end   = s[s.length() - 1];
if (s.find("e") != string::npos) {
    string first = s.substr(0, 5);
    string last  = s.substr(6);
}
if (s == "Sonia Sotomayor") {
    cout << "John Roberts" << endl;
}

```

```

"""          Python Version          """
s = "Elena Kagan"
s += ", joined " + str(2010)
start = s[0]
end   = s[-1]
if 'e' in s:
    first = s[0:5]
    last  = s[6:]
if s == "Sonia Sotomayor":

```

You can select an individual character out of a string by using square brackets. Indices start at zero.

C++ has different types for individual characters (**char**) and for strings of zero or more characters (**string**). Check Chapter 1.5 of the textbook for details.

```

/*          Java Version          */
String s = "Elena Kagan";
s += ", joined " + 2010;
char start = s.charAt(0);
char end   = s.charAt(s.length() - 1);
if (s.indexOf("e") != -1) {
    String first = s.substring(0, 5);
    String last  = s.substring(6);
}
if (s.equals("Sonia Sotomayor")) {
    System.out.println("John Roberts");
}

```



```

/*          C++ Version          */
string s = "Elena Kagan";
s += ", joined " + to_string(2010);

char start = s[0];
char end    = s[s.length() - 1];
if (s.find("e") != string::npos) {
    string first = s.substr(0, 5);
    string last  = s.substr(6);
}

if (s == "Sonia Sotomayor") {
    cout << "John Roberts" << endl;
}

```

```

"""          Python Version          """
s = "Elena Kagan"
s += ", joined " + str(2010)

start = s[0]
end    = s[-1]
if 'e' in s:
    first = s[0:5]
    last  = s[6:]

if s == "Sonia Sotomayor":
    print("John Roberts")

```

```

/*          Java Version          */
String s = "Elena Kagan";
s += ", joined " + 2010;

char start = s.charAt(0);
char end    = s.charAt(s.length() - 1);
if (s.indexOf("e") != -1) {
    String first = s.substring(0, 5);
    String last  = s.substring(6);
}

if (s.equals("Sonia Sotomayor")) {
    System.out.println("John Roberts");
}

```

C++ doesn't support negative array indices the way that Python does. You can pick the last character of the string by getting its length and subtracting one.

```

let last = s.substring(6);
}

if (s === "Sonia Sotomayor") {
    console.log("John Roberts");
}

```

```

/*          C++ Version          */
string s = "Elena Kagan";
s += ", joined " + to_string(2010);

char start = s[0];
char end   = s[s.length() - 1];

if (s.find("e") != string::npos) {
    string first = s.substr(0, 5);
    string last  = s.substr(6);
}

if (s == "Sonia Sotomayor") {
    cout << "John Roberts" << endl;
}

```

```

"""          Python Version          """
s = "Elena Kagan"
s += ", joined " + str(2010)

start = s[0]
end   = s[-1]

if 'e' in s:
    first = s[0:5]
    last  = s[6:]

if s == "Sonia Sotomayor":
    print("John Roberts")

```

```

/*          Java Version          */
String s = "Elena Kagan";
s += ", joined " + 2010;

char start = s.charAt(0);
char end   = s.charAt(s.length() - 1);

if (s.indexOf("e") != -1) {
    String first = s.substring(0, 5);
    String last  = s.substring(6);
}

if (s.equals("Sonia Sotomayor")) {
    System.out.println("John Roberts");
}

```

The find member function returns the index of the given pattern if it exists, and the verbosely-named constant `string::npos` otherwise. This pattern kinda sorta is like the `in` keyword from Python.

```

console.log("John Roberts");
}

```

```
/*          C++ Version          */
string s = "Elena Kagan";
s += ", joined " + to_string(2010);
char start = s[0];
char end    = s[s.length() - 1];
if (s.find("e") != string::npos) {
    string first = s.substr(0, 5);
    string last  = s.substr(6);
}
if (s == "Sonia Sotomayor") {
    cout << "John Roberts" << endl;
}
```

```
"""          Python Version          """
s = "Elena Kagan"
s += ", joined " + str(2010)
start = s[0]
end    = s[-1]
if 'e' in s:
    first = s[0:5]
    last  = s[6:]
if s == "Sonia Sotomayor":
    print("John Roberts")
```

```
/*          Java Version          */
String s = "Elena Kagan";
s += ", joined " + 2010;
char start = s.charAt(0);
char end    = s.charAt(s.length() - 1);
if (s.indexOf("e") != -1) {
    String first = s.substring(0, 5);
    String last  = s.substring(6);
}
if (s.equals("Sonia Sotomayor")) {
    System.out.println("John Roberts");
}
```

```
//          JavaScript Version
```

You can get substrings by using the `.substr` member function. If you give two parameters, the first is a start index, and the second is a length, not an end index.

```
console.log("John Roberts");
}
```

```
/*          C++ Version          */
string s = "Elena Kagan";
s += ", joined " + to_string(2010);

char start = s[0];
char end    = s[s.length() - 1];

if (s.find("e") != string::npos) {
    string first = s.substr(0, 5);
    string last  = s.substr(6);
}

if (s == "Sonia Sotomayor") {
    cout << "John Roberts" << endl;
}
```

```
"""          Python Version          """
s = "Elena Kagan"
s += ", joined " + str(2010)

start = s[0]
end    = s[-1]

if 'e' in s:
    first = s[0:5]
    last  = s[6:]

if s == "Sonia Sotomayor":
    print("John Roberts")
```

```
/*          Java Version          */
String s = "Elena Kagan";
s += ", joined " + 2010;

char start = s.charAt(0);
char end    = s.charAt(s.length() - 1);

if (s.indexOf("e") != -1) {
    String first = s.substring(0, 5);
    String last  = s.substring(6);
}

if (s.equals("Sonia Sotomayor")) {
    System.out.println("John Roberts");
}
```

```
//          JavaScript Version
```

You can compare strings for equality using `==`. If you're coming from Python, great! This will feel normal. If you're coming from Java, hopefully this will be a welcome relief.

```
/*          C++ Version          */
string s = "Elena Kagan";
s += ", joined " + to_string(2010);
char start = s[0];
char end   = s[s.length() - 1];
if (s.find("e") != string::npos) {
    string first = s.substr(0, 5);
    string last  = s.substr(6);
}
if (s == "Sonia Sotomayor") {
    cout << "John Roberts" << endl;
}
```

```
"""          Python Version          """
s = "Elena Kagan"
s += ", joined " + str(2010)
start = s[0]
end   = s[-1]
if 'e' in s:
    first = s[0:5]
    last  = s[6:]

if s == "Sonia Sotomayor":
    print("John Roberts")
```

```
/*          Java Version          */
String s = "Elena Kagan";
s += ", joined " + 2010;
char start = s.charAt(0);
char end   = s.charAt(s.length() - 1);
if (s.indexOf("e") != -1) {
    String first = s.substring(0, 5);
    String last  = s.substring(6);
}
if (s.equals("Sonia Sotomayor")) {
    System.out.println("John Roberts");
}
```

```
/*          JavaScript Version          */
s = "Elena Kagan";
s += ", joined " + 2010;
start = s[0];
end   = s[s.length - 1];
if (s.indexOf("e") != -1) {
    let first = s.substring(0, 5);
    let last  = s.substring(6);
}
if (s === "Sonia Sotomayor") {
    console.log("John Roberts");
}
```

You can print strings the same way you print anything else.

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.
- Variables in C++ represent the actual values they describe, rather than pointers or references to those values.
- 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.
- C++ strings support most of the “stringy” operations you expect from other programming languages, and have their share of quirks.

Your Action Items

- ***Read Chapter 2 and Chapter 3.***
 - We're still easing into C++. These chapters talk about the basics and the mechanics of function call and return.
- ***Sign up for a Discussion Section.***
 - The link goes out tomorrow afternoon.
- ***Work on Assignment 0.***
 - Just over a third of you are already done!
Exciting!

Next Time

- ***Recursion***

- Solving problems by solving smaller copies of the same problem.

- ***Recursion***

- Solving problems by solving smaller copies of the same problem.

- ***Recursion***

- Solving problems by solving smaller copies of the same problem.

- ***Recursion***

- Solving problems by solving smaller copies of the same problem.

- ***Recursion***

- ...

Appendix: Example: Cyclic Shifts

Example: Cyclic Shifts

Cyclic Equality

- You can *cyclically shift* a word by moving a block of characters from the back of the word to the front.

T	A	P	A	S
---	---	---	---	---

- Sometimes, the result is a new word.

Cyclic Equality

- You can *cyclically shift* a word by moving a block of characters from the back of the word to the front.

O	F	F	H	A	N	D
---	---	---	---	---	---	---

- Sometimes, the result is a new word.

Cyclic Equality

- You can *cyclically shift* a word by moving a block of characters from the back of the word to the front.

S	W	O	R	D	P	L	A	Y
---	---	---	---	---	---	---	---	---

- Sometimes, the result is a new word.

Our Strategy

Check if one == two.

Cycle the string.

Check if one == two.

Cycle the string.

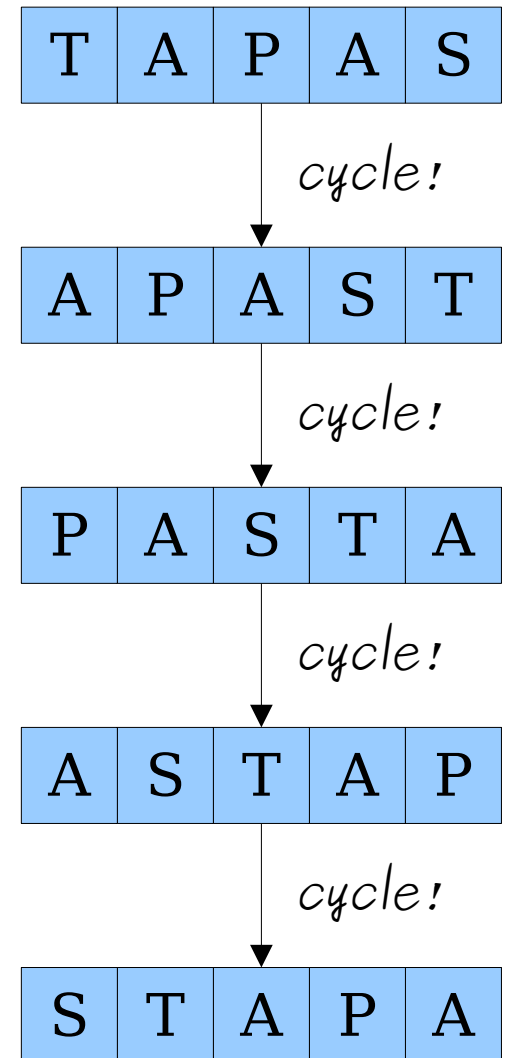
Check if one == two.

Cycle the string.

Check if one == two.

Cycle the string.

Check if one == two.



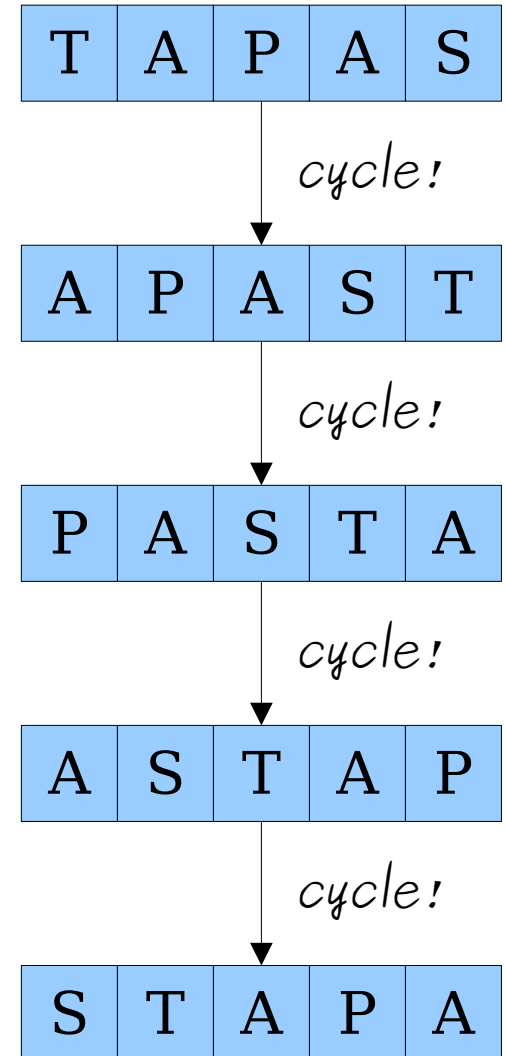
Our Strategy

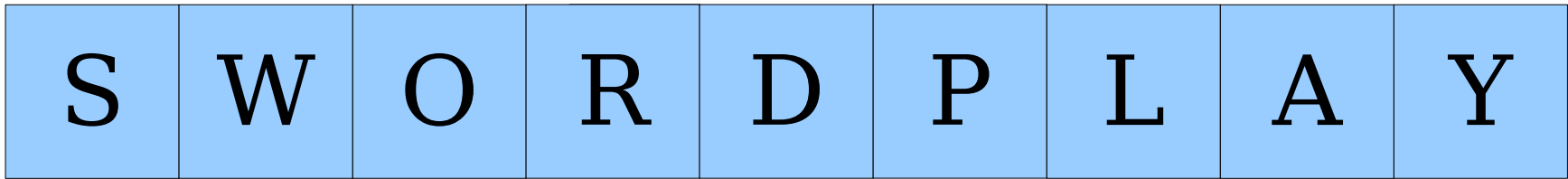
Do these steps
in a loop

Check if one == two.
Cycle the string.
Check if one == two.
Cycle the string.
Check if one == two.
Cycle the string.
Check if one == two.
Cycle the string.

Handle this by
itself.

Check if one == two.





`str[0]`

`str.substr(1)`

W	O	R	D	P	L	A	Y	S
---	---	---	---	---	---	---	---	---

`str.substr(1) + str[0]`

Tracing Function Calls

```
int main() {  
    string one = getLine("First: ");  
    string two = getLine("Second: ");  
  
    if (areCyclicallyEqual(one, two)) {  
        cout << "Yep!" << endl;  
    } else {  
        cout << "Nah." << endl;  
    }  
  
    return 0;  
}
```

Tracing Function Calls

```
int main() {  
    string one = getLine("First: ");  
    string two = getLine("Second: ");  
  
    if (areCyclicallyEqual(one, two)) {  
        cout << "Yep!" << endl;  
    } else {  
        cout << "Nah." << endl;  
    }  
  
    return 0;  
}
```

Tracing Function Calls

```
int main() {  
    string one = getLine("First: ");  
    string two = getLine("Second: ");  
  
    if (areCyclicallyEqual(one, two)) {  
        cout << "Yep!" << endl;  
    } else {  
        cout << "Nah." << endl;  
    }  
  
    return 0;  
}
```

pasta

one

Tracing Function Calls

```
int main() {  
    string one = getLine("First: ");  
    string two = getLine("Second: ");  
  
    if (areCyclicallyEqual(one, two)) {  
        cout << "Yep!" << endl;  
    } else {  
        cout << "Nah." << endl;  
    }  
  
    return 0;  
}
```

pasta

one

Tracing Function Calls

```
int main() {  
    string one = getLine("First: ");  
    string two = getLine("Second: ");  
  
    if (areCyclicallyEqual(one, two)) {  
        cout << "Yep!" << endl;  
    } else {  
        cout << "Nah." << endl;  
    }  
  
    return 0;  
}
```

pasta

tapas

one

two

String variables in C++ represent actual strings, rather than pointers to strings stored elsewhere.

Tracing Function Calls

```
int main() {  
    string one = getLine("First: ");  
    string two = getLine("Second: ");
```

```
    if (areCyclicallyEqual(one, two)) {  
        cout << "Yep!" << endl;  
    } else {  
        cout << "Nah." << endl;  
    }
```

```
    return 0;
```

```
}
```

pasta

one

tapas

two

Tracing Function Calls

```
int main() {
```

pasta

tapas

```
bool areCyclicallyEqual(string one,  
                        string two) {
```

pasta

tapas

one

two

```
for (int i = 1; i < one.length(); i++) {  
    if (one == two) {  
        return true;  
    }
```

```
two = two.substr(1) + two[0];
```

```
}
```

```
return one == two;
```

```
}
```


Tracing Function Calls

```
int main() {
```

```
    bool areCyclicallyEqual(string one,  
                             string two) {
```

```
        for (int i = 1; i < one.length(); i++) {  
            if (one == two) {  
                return true;  
            }  
        }
```

```
        two = two.substr(1) + two[0];  
    }
```

```
    return one == two;
```

```
}
```

pasta

tapas

pasta

tapas

one

two

We get our own
fresh copies of
one and two.

Tracing Function Calls

```
int main() {
```

pasta

tapas

```
bool areCyclicallyEqual(string one,  
                        string two) {
```

pasta

tapas

one

two

```
for (int i = 1; i < one.length(); i++) {  
    if (one == two) {  
        return true;  
    }
```

```
two = two.substr(1) + two[0];
```

```
}
```

```
return one == two;
```

```
}
```

Tracing Function Calls

```
int main() {
```

pasta

tapas

```
bool areCyclicallyEqual(string one,  
                        string two) {
```

pasta

tapas

one

two

```
for (int i = 1; i < one.length(); i++) {
```

```
    if (one == two) {
```

```
        return true;
```

```
    }
```

```
    two = two.substr(1) + two[0];
```

```
}
```

```
return one == two;
```

```
}
```

Tracing Function Calls

```
int main() {
```

pasta

tapas

```
bool areCyclicallyEqual(string one,  
                        string two) {
```

```
for (int i = 1; i < one.length(); i++) {
```

```
if (one == two) {
```

```
return true;
```

```
}
```

```
two = two.substr(1) + two[0];
```

```
}
```

```
return one == two;
```

```
}
```

pasta

tapas

one

two

Tracing Function Calls

```
int main() {
```

pasta

tapas

```
bool areCyclicallyEqual(string one,  
                        string two) {
```

pasta

tapas

one

two

```
for (int i = 1; i < one.length(); i++) {  
    if (one == two) {  
        return true;  
    }  
}
```

```
two = two.substr(1) + two[0];
```

```
}
```

```
return one == two;
```

```
}
```

Tracing Function Calls

```
int main() {
```

pasta

tapas

```
bool areCyclicallyEqual(string one,  
                        string two) {
```

```
for (int i = 1; i < one.length(); i++) {  
    if (one == two) {  
        return true;  
    }
```

```
two = two.substr(1) + two[0];
```

```
return one == two;
```

pasta

apast

one

two

Tracing Function Calls

```
int main() {
```

pasta

tapas

```
bool areCyclicallyEqual(string one,  
                        string two) {
```

```
for (int i = 1; i < one.length(); i++) {
```

```
if (one == two) {
```

```
return true;
```

```
}
```

```
two = two.substr(1) + two[0];
```

```
}
```

```
return one == two;
```

```
}
```

pasta

apast

one

two

Tracing Function Calls

```
int main() {
```

pasta

tapas

```
bool areCyclicallyEqual(string one,  
                        string two) {
```

```
for (int i = 1; i < one.length(); i++) {
```

```
if (one == two) {
```

```
return true;
```

```
}
```

```
two = two.substr(1) + two[0];
```

```
}
```

```
return one == two;
```

```
}
```

pasta

apast

one

two

Tracing Function Calls

```
int main() {
```

pasta

tapas

```
bool areCyclicallyEqual(string one,  
                        string two) {
```

```
for (int i = 1; i < one.length(); i++) {  
    if (one == two) {  
        return true;  
    }
```

```
two = two.substr(1) + two[0];
```

```
return one == two;
```

pasta

apast

one

two

Tracing Function Calls

```
int main() {
```

pasta

tapas

```
bool areCyclicallyEqual(string one,  
                        string two) {
```

```
    for (int i = 1; i < one.length(); i++) {  
        if (one == two) {  
            return true;  
        }  
    }
```

```
    two = two.substr(1) + two[0];
```

```
    return one == two;
```

pasta

pasta

one

two

Tracing Function Calls

```
int main() {
```

pasta

tapas

```
bool areCyclicallyEqual(string one,  
                        string two) {
```

```
for (int i = 1; i < one.length(); i++) {
```

```
if (one == two) {
```

```
return true;
```

```
}
```

```
two = two.substr(1) + two[0];
```

```
}
```

```
return one == two;
```

```
}
```

pasta

pasta

one

two

Tracing Function Calls

```
int main() {
```

pasta

tapas

```
bool areCyclicallyEqual(string one,  
                        string two) {
```

```
for (int i = 1; i < one.length(); i++) {
```

```
if (one == two) {
```

```
return true;
```

```
}
```

```
two = two.substr(1) + two[0];
```

```
}
```

```
return one == two;
```

```
}
```

pasta

pasta

one

two

Tracing Function Calls

```
int main() {
```

pasta

tapas

```
bool areCyclicallyEqual(string one,  
                        string two) {
```

```
    for (int i = 1; i < one.length(); i++) {
```

```
        if (one == two) {
```

```
            return true;
```

```
        }
```

```
        two = two.substr(1) + two[0];
```

```
    }
```

```
    return one == two;
```

```
}
```

pasta

pasta

one

two

Tracing Function Calls

```
int main() {
```

pasta

tapas

```
bool areCyclicallyEqual(string one,  
                        string two) {
```

pasta

pasta

one

two

```
for (int i = 1; i < one.length(); i++) {
```

```
if (one == two) {
```

```
return true;
```

```
}
```

```
two = two.substr(1) + two[0];
```

```
}
```

```
return one == two;
```

```
}
```

Tracing Function Calls

```
int main() {  
    string one = getLine("First: ");  
    string two = getLine("Second: ");  
  
    if (areCyclicallyEqual(one, two)) {  
        cout << "Yep!" << endl;  
    } else {  
        cout << "Nah." << endl;  
    }  
  
    return 0;  
}
```

pasta

one

tapas

two

Tracing Function Calls

```
int main() {  
    string one = getLine("First: ");  
    string two = getLine("Second: ");  
  
    if (areCyclicallyEqual(one, two)) {  
        cout << "Yep!" << endl;  
    } else {  
        cout << "Nah." << endl;  
    }  
  
    return 0;  
}
```

pasta

tapas

one

two

Notice that our original copies of one and two are unmodified.

Tracing Function Calls

```
int main() {  
    string one = getLine("First: ");  
    string two = getLine("Second: ");  
  
    if (areCyclicallyEqual(one, two)) {  
        cout << "Yep!" << endl;  
    } else {  
        cout << "Nah." << endl;  
    }  
  
    return 0;  
}
```

pasta

tapas

one

two

Notice that our original copies of one and two are unmodified.