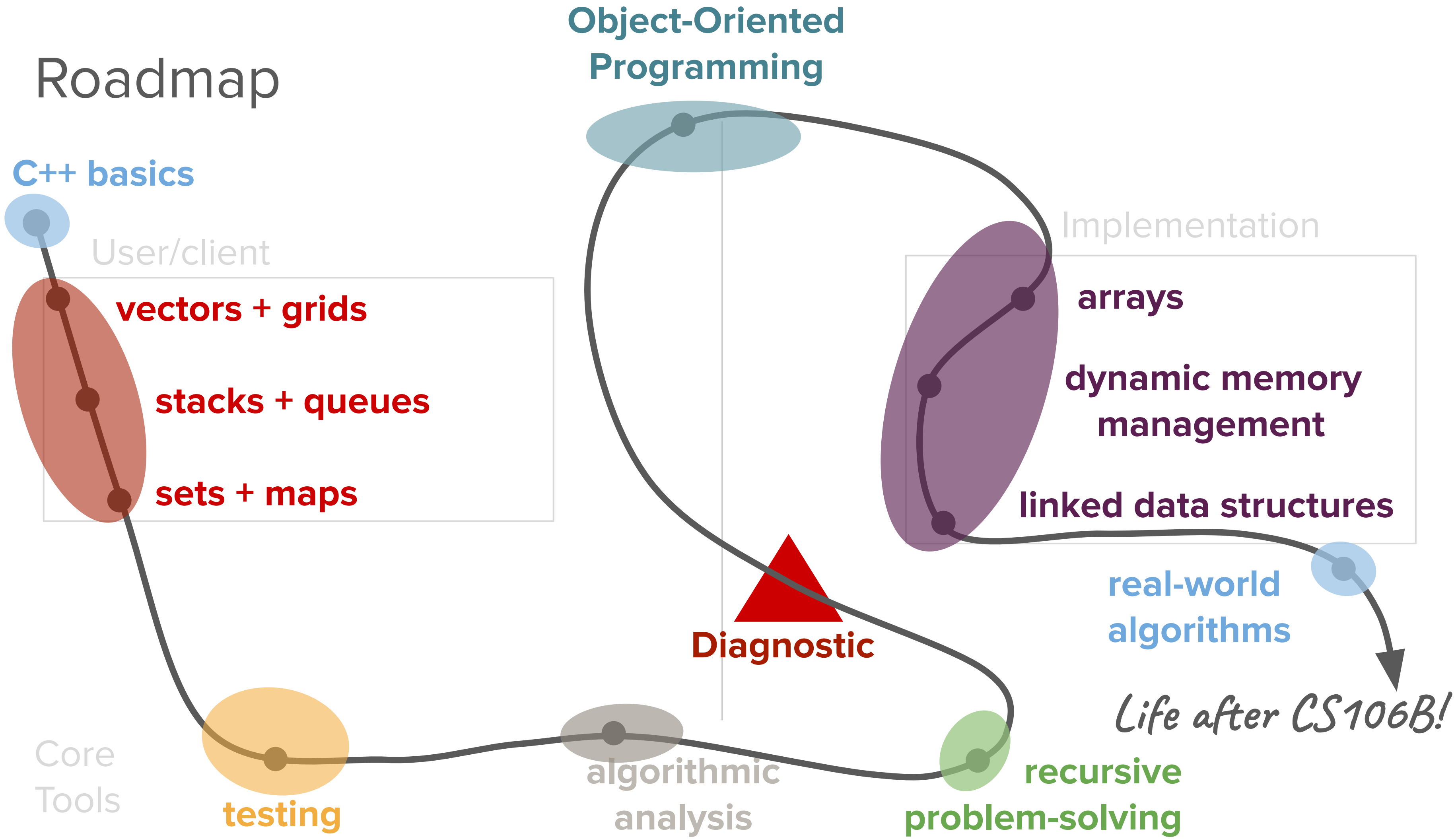
Strings and Testing

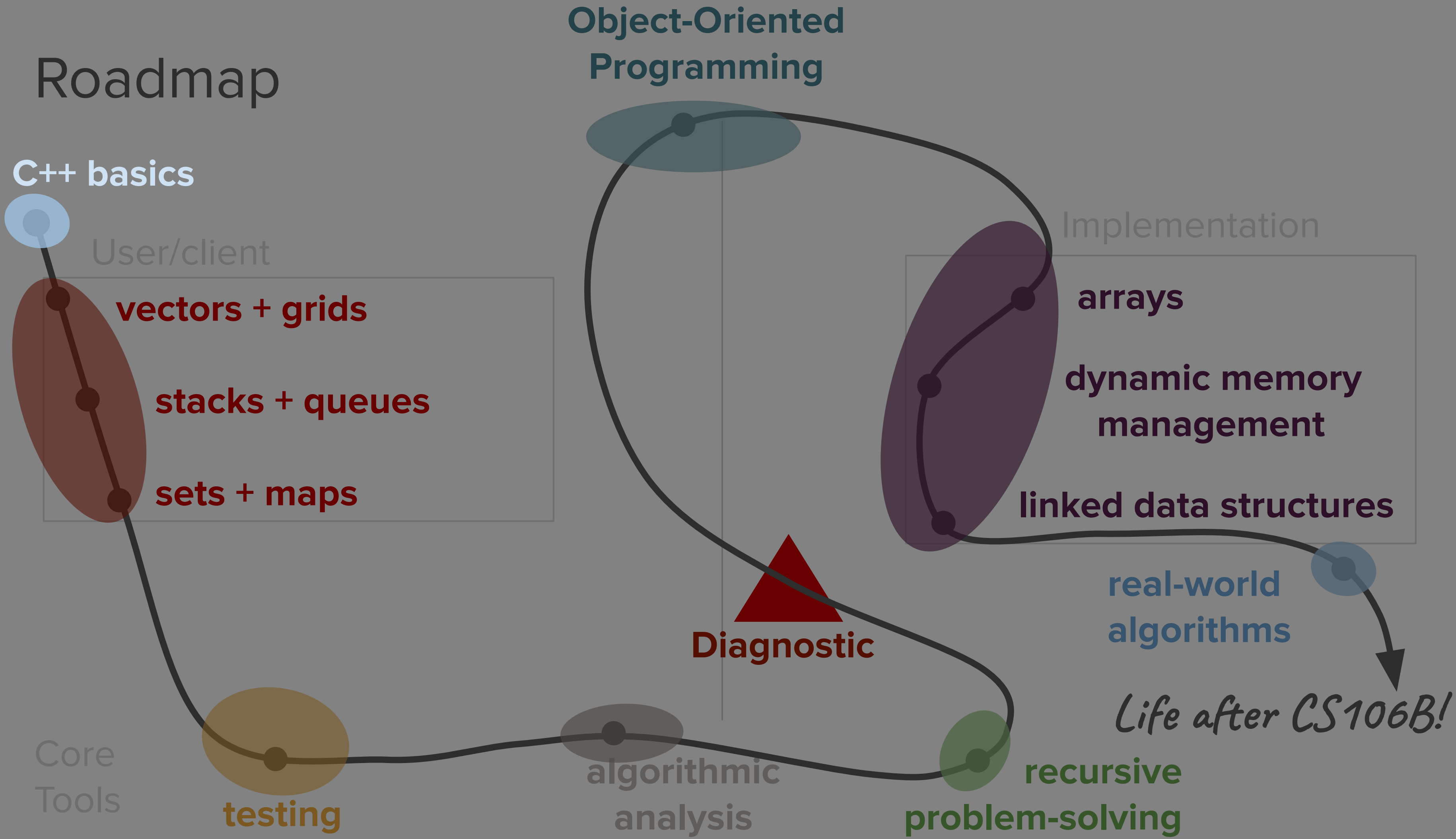
What strategies have resulted in effective breakout rooms for you during online learning?

(put your answers the chat)

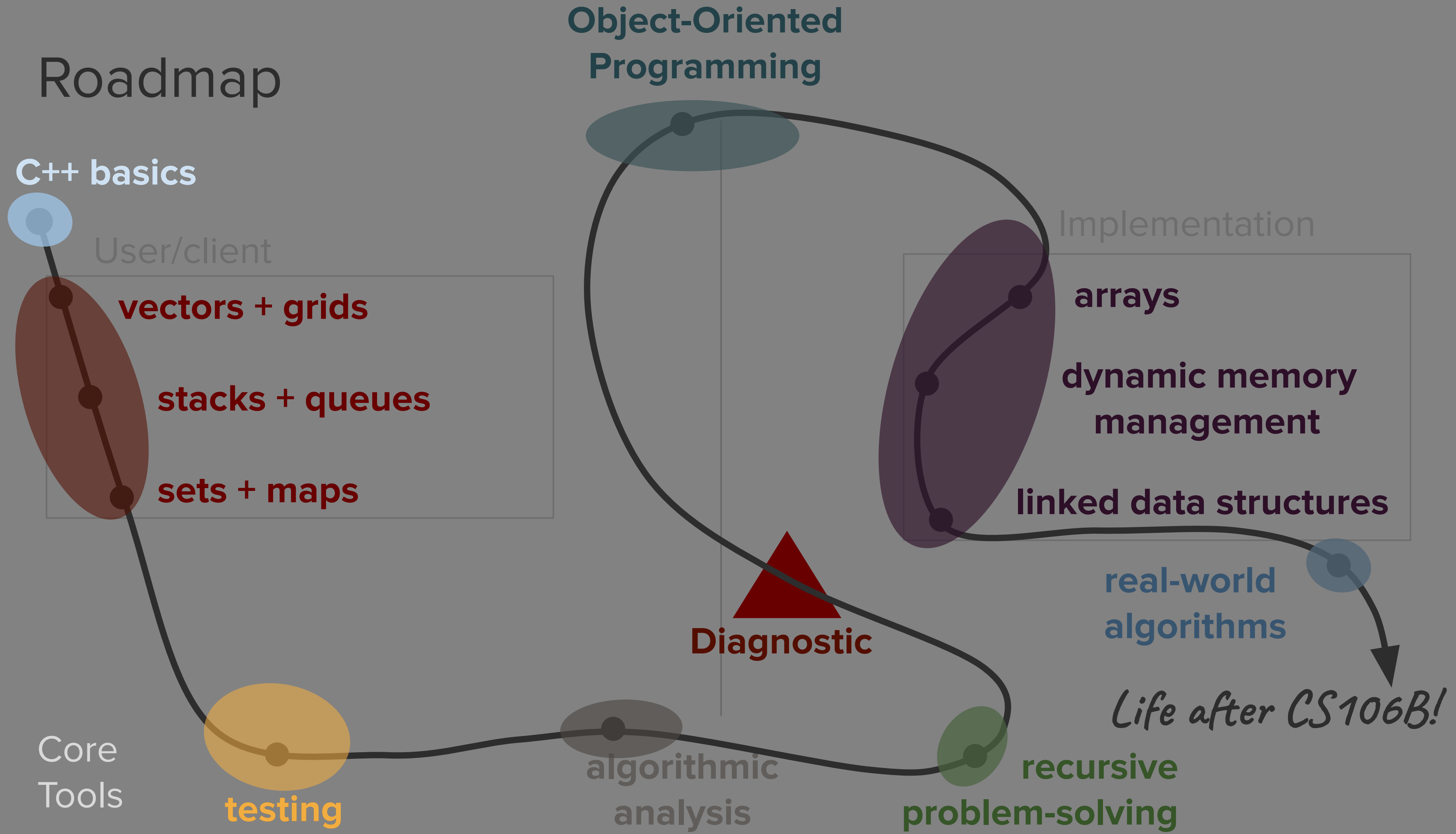
Roadmap


Object-Oriented Programming
C++ basics
User/client
vectors + grids
stacks + queues
sets + maps
Implementation
arrays
dynamic memory management
linked data structures
real-world algorithms
Diagnostic
Life after CS106B!
Core Tools
testing
algorithmic analysis
recursive problem-solving

Roadmap


Object-Oriented Programming
C++ basics
User/client
vectors + grids
stacks + queues
sets + maps
Implementation
arrays
dynamic memory management
linked data structures
real-world algorithms
Diagnostic
Life after CS106B!
Core Tools
testing
algorithmic analysis
recursive problem-solving

Roadmap


Object-Oriented Programming
C++ basics
User/client
vectors + grids
stacks + queues
sets + maps
Implementation
arrays
dynamic memory management
linked data structures
real-world algorithms
Diagnostic
Life after CS106B!
Core Tools
testing
algorithmic analysis
recursive problem-solving

Today's questions

C++ survey questions + review

What's special about strings in C++?

How do we test code in CS106B?

What's next?

C++ survey questions

Questions you asked!

We looked over your questions in the C++ survey...

- Many questions/topics Nick already covered in Wednesday’s lecture.

- I’ll review some of the most high-demand topics today.
  .
- Other questions will get covered in upcoming lectures.

- Some specific questions + more supplementary questions will get answered in an upcoming Ed post.

Questions that will be covered in upcoming lectures

- Data Structures
  - No questions in mind at the moment. I guess it would just be helpful to go over **dictionaries**
  - What roles do **dictionaries** play in C++?
  - How do you create **lists/dictionaries** through C++ like we learned in CS 106A for python?
  - How the **data types** we know and love from python translate to C++.
  - Boolean expressions and useful standard mathematical functions that we have at our disposal. Also what sorts of iterable types exist (are there **arrays**? **lists**? **tuples**? etc)
  - **Vectors** in C++ and how they're different from numpy
- Object Oriented Programming
  - What are **classes**?
  - C++ **Class**, **private** and **public** sections
- Low Level C++
  - newer C++ features, **pointers**

Questions we’ll answer in the "Bonus" Ed Post

- Language Comparisons and Transitions (C vs. C++ vs. Python vs. many more)
  - How the pros and cons of C++ can be related to where it's used in industry
  - A brief comparison (field of application, shrinks and weaknesses) between Python and C++
  - Some advantages of C++ over Python; why some industries rely heavily on it while others don't
  - What are the most important functional/syntactical differences when coding in C++ vs C?
  - I took CS106A [...] in Java, so I would love a little clarity regarding how C++ is similar to Java
  - General comments on differences between Python/MATLAB.
  - What are the main misconceptions on C++? What are common mistakes of going from one language to another? What are key things to look out for?
- Miscellaneous
  - Can you review function prototypes and why they are used?
  - Can you talk about include statements/importing libraries in detail?
  - How to be effective in using online resources or the textbook while working on assignments.
  - How can computer code be used to communicate with machines?

Questions you asked: Utility functions

- “Utility functions” is just a term we use to refer to commonly used functions that are often built-into the C++ language or are similar in functionality to built-in functions in Python.
    - It’s not a formal term, just a useful catch-all phrase. We don’t expect you to identify what functions are/are not “utility” functions.

- We’ll be learning about string-specific utility functions today! (some built-in and some not)

- We’ll cover important ones in class or mention them in the useful tips section of assignment handouts.

Questions you asked: Style + Formatting

- We prefer to put the opening curly brace at the end of the line where a function/control structure begins. For example:

```
if (blah blah) {
    ...
}
```

  Some people also like putting it on its own line – either is fine as long as you're consistent! **But you should indent by one level everything within the curly braces**.

- Read through the CS106B style guide on the website!

Questions you asked: Function prototypes

Does the compiler read functions from the top of the script to the bottom, or in order of when they are called?

Top to bottom – this is why function prototypes + forward declarations are useful!

Example function prototype:

```
    returnType functionName(varType parameter1, varType parameter2, ...);
```

This usually go toward the top of the file and allow you to use functions above the place where they're fully defined. We'll go into more depth on this later (when we talk about classes) so don't worry about them for now!

Questions you asked: void + return types

```
    returnType functionName(varType parameter1, varType parameter2, ...);
```

The returnType indicates the type of data your function is outputting (e.g. **`int`**, **`double`**, **`string`**, etc.)

We use **`void`** to indicate that your function is not outputting, or *returning*, anything *back to the caller function* (i.e. where it was called).

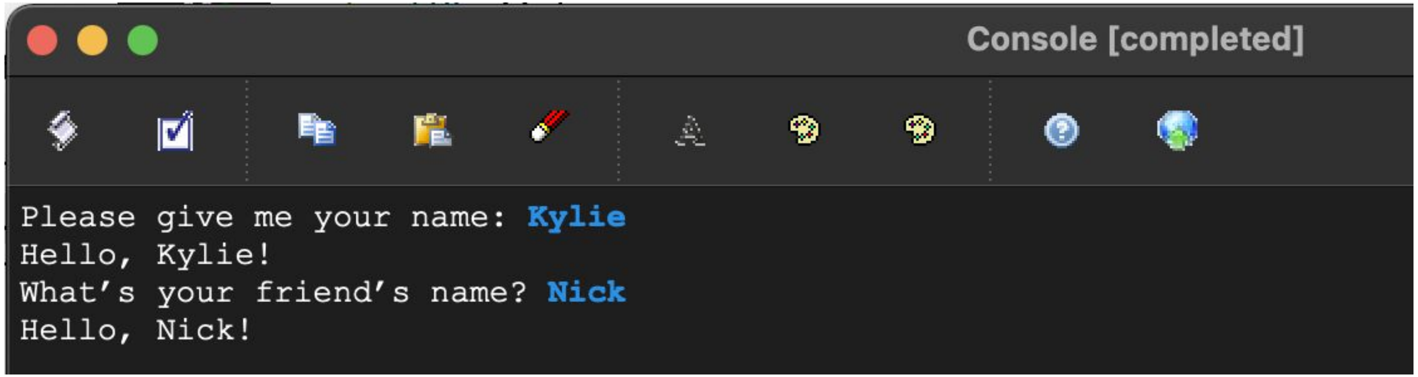
For example, functions that are used to decompose out logic for just printing something on the screen might not need to return anything to main.

Questions you asked: void + return types

For example, functions that are used to decompose out logic for just printing something on the screen might not need to return anything to main.

```
void printName(string name) {
    cout << “Hello, ” << name << “!” << endl;
}

int main() {
    string userName = getLine(“Please give me your name: ”);
    printName(username);
    string friendName = getLine(“What’s your friend’s name? ”);
    printName(friendName);
}
```

Questions you asked: Console output



```
Please give me your name: Kylie
Hello, Kylie!
What's your friend's name? Nick
Hello, Nick!
```

When concatenating strings with variables is there a need to use '+' signs? How are these understood and differentiated if only using '<<' signs?

Great question! The `<<` signs are actually only for console output. You can't use them for combining strings to store in a variable. For concatenating strings together and storing them, you need to use the `+` operator.

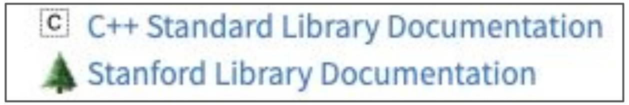
Questions you asked: Transitioning to C++

What would you recommend is the best way to help the transition from [insert language here] to C++. Are there any resources that can expedite the process?

What's the best way to find documentation about standard/Stanford C++ libraries?

As we go, we'll talk about common C++ misconceptions coming from other languages. The Ed post will also go into more detail for specific languages and their differences.

Under "Quick Links" on the CS106B website homepage:



C++ Standard Library Documentation
Stanford Library Documentation

Review: Loops + Data Types

Loops

`while` loops

- Loops allow you to repeat the execution of a certain block of code multiple times

`while` loops

- Loops allow you to repeat the execution of a certain block of code multiple times
- `while` loops are great when you want to continue executing something until a certain condition is met and you don't know exactly how many times you want to iterate for

`while` loops

- Loops allow you to repeat the execution of a certain block of code multiple times
- `while` loops are great when you want to continue executing something until a certain condition is met and you don't know exactly how many times you want to iterate for

```
while (expression) {
  statement;
  statement;
  ...
}
```

Execution continues until expression evaluates to false

`while` loops

- Loops allow you to repeat the execution of a certain block of code multiple times
- **`while`** loops are great when you want to continue executing something until a certain condition is met and you don't know exactly how many times you want to iterate for

```
while (expression) {
  statement;
  statement;
  ...
}
```

```
int i = 0;
while (i < 5) {
  cout << i << endl;
  i++;
}
```

```
Output:
0
1
2
3
4
```

`for` loops

- **`for`** loops are great when you have a known, fixed number of times that you want to execute a block of code

`for` loops

- **`for`** loops are great when you have a known, fixed number of times that you want to execute a block of code
- **`for`** loop syntax in C++ can look a little strange, let's investigate!

`for` loops

- `for` loops are great when you have a known, fixed number of times that you want to execute a block of code

```
for (initializationStatement; testExpression; updateStatement) {
  statement;
  statement;
  ...
}
```

`for` loops

- `for` loops are great when you have a known, fixed number of times that you want to execute a block of code

```
for (initializationStatement; testExpression; updateStatement) {
    statement;
    statement;
    ...
}
```

The `initializationStatement` happens at the beginning of the loop, and initializes a variable.

E.g., `int i = 0`.

`for` loops

- **`for`** loops are great when you have a known, fixed number of times that you want to execute a block of code

```
for (initializationStatement; testExpression; updateStatement) {
    statement;
    statement;
    ...
}
```

The **`testExpression`** is evaluated initially, and after each run through the loop, and if it is **`true`**, the loop continues for another iteration.

E.g., **`i < 3`**.

`for` loops

- `for` loops are great when you have a known, fixed number of times that you want to execute a block of code

```
for (initializationStatement; testExpression; updateStatement) {
    statement;
    statement;
    ...
}
```

The `updateStatement` happens after each loop, but *before* `testExpression` is evaluated.

E.g., **i++**.

`for` loops

- `for` loops are great when you have a known, fixed number of times that you want to execute a block of code

```
for (initializationStatement; testExpression; updateStatement) {
    statement;
    statement;
    ...
}
```

```
for (int i = 0; i < 3; i++) {
    cout << i << endl;
}
```

`for` loops

- `for` loops are great when you have a known, fixed number of times that you want to execute a block of code

```
for (initializationStatement; testExpression; updateStatement) {
  statement;
  statement;
  ...
}
```

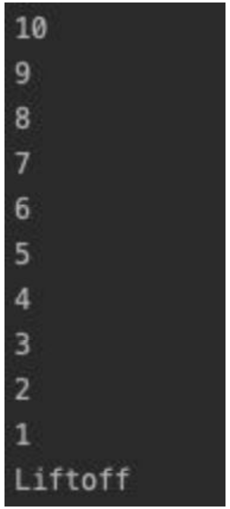
⇩

```
for (int i = 0; i < 3; i++) {
  cout << i << endl;
}
```

```
Output:
0
1
2
```

Putting it together with `spaceship.cpp`

Write a program that prints out the calls for a spaceship that is about to launch. Countdown the numbers from 10 to 1 and then print "Liftoff."



```
10
9
8
7
6
5
4
3
2
1
Liftoff
```

```python
def main():
    for i in range(10, 0, -1):
        print(i)
    print("Liftoff")

if __name__ == "__main__":
    main()
```

Python

```cpp
#include <iostream>
using namespace std;

int main() {
    /* TODO: Your code goes here! */

    return 0;
}
```

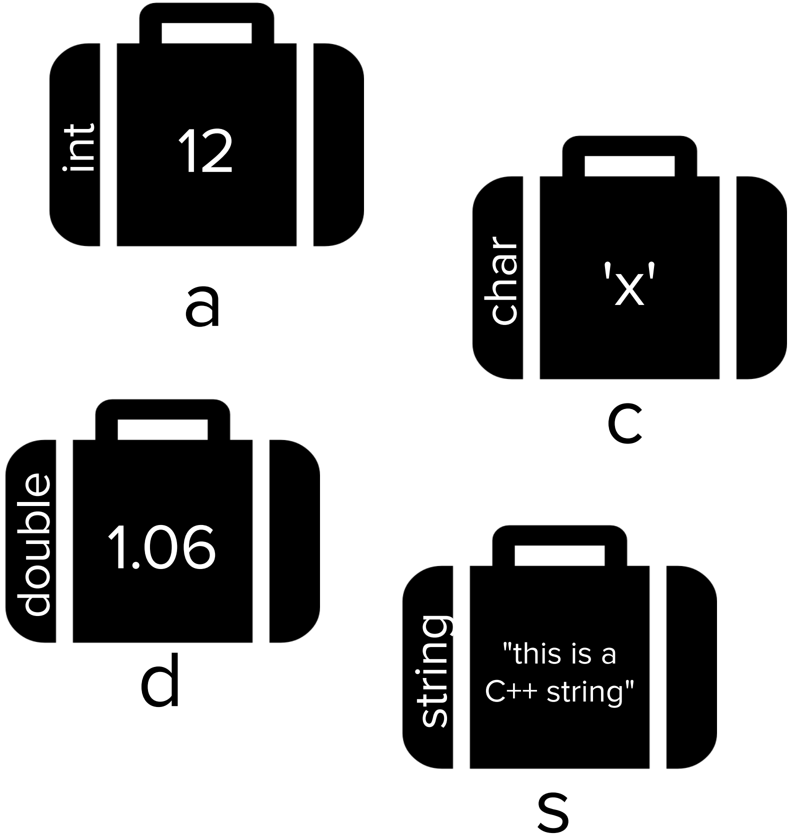
C++

Ed activity

(workspaces)

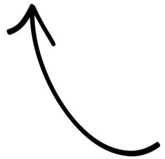
Recall data types...

```
int num = 5; // declare a new integer var

char letter = 'x'; // b is a char ("character")

double decimal = 1.06; // d is a double, a type
used to represent decimal numbers

string sentence = "this is a C++ string";

bool isRaining = false;
```

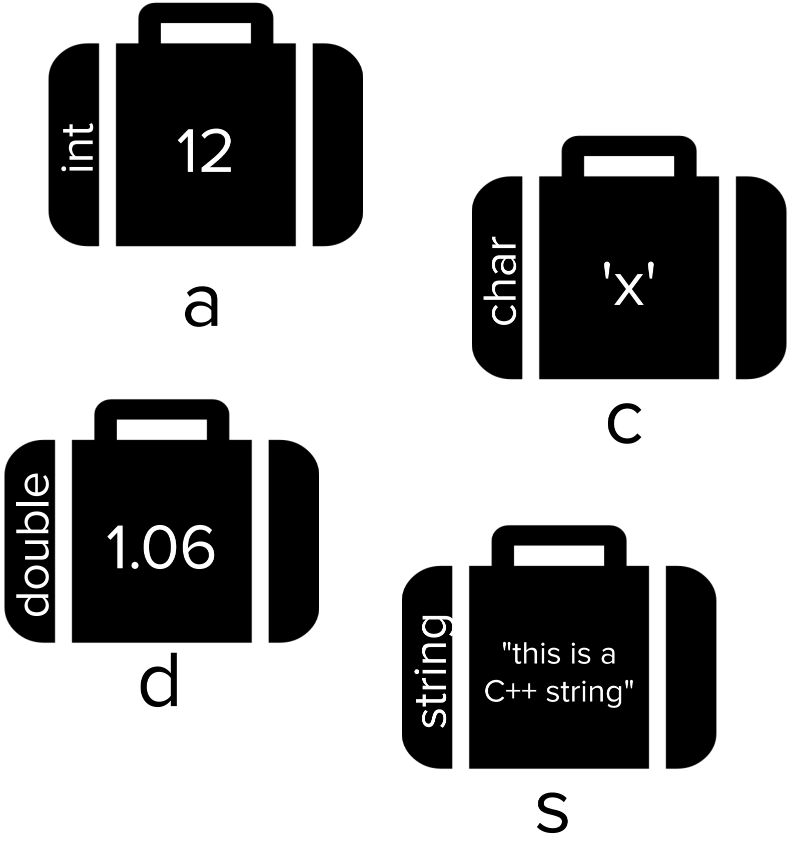


Recall data types...

```
int num = 5; // declare a new integer var

char letter = 'x'; // b is a char ("character")

double decimal = 1.06; // d is a double, a type
used to represent decimal numbers

string sentence = "this is a C++ string";

bool isRaining = false;
```

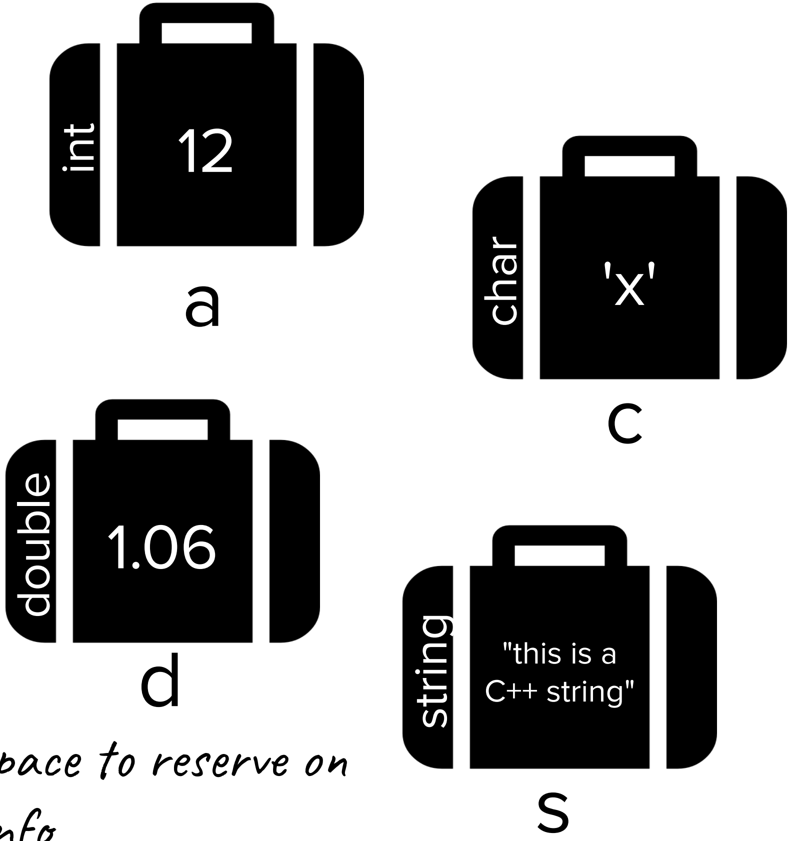


We forgot to mention booleans on Wednesday!



Recall data types...

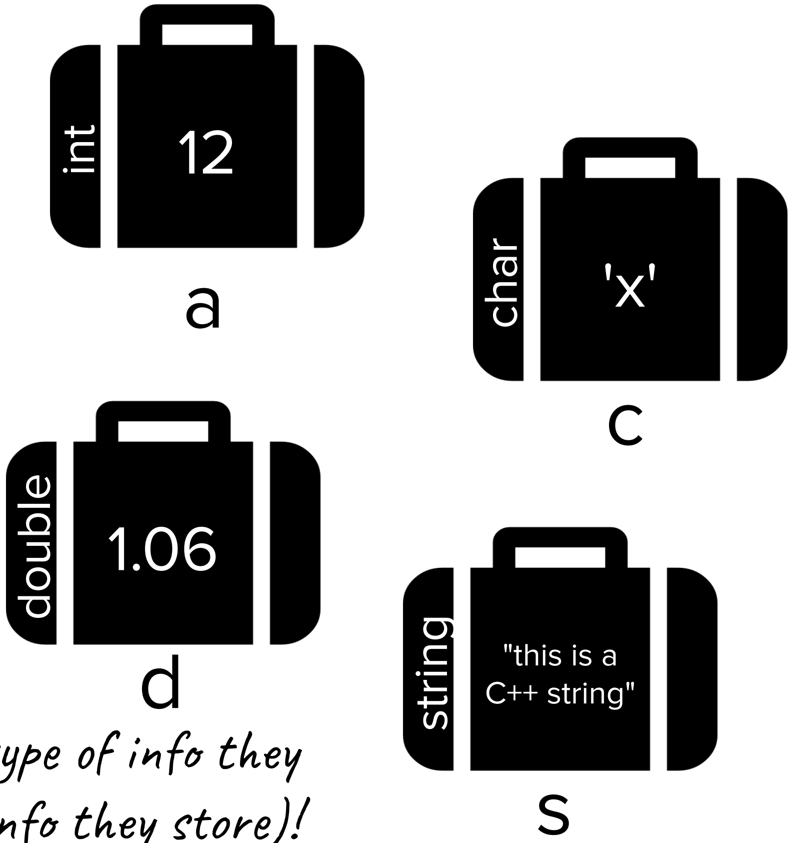
```
int num = 5; // declare a new integer var

char letter = 'x'; // b is a char ("character")

double decimal = 1.06; // d is a double, a type
used to represent decimal numbers

string sentence = "this is a C++ string";

bool isRaining = false;
```



Note: Types are C++'s way of knowing how much space to reserve on your computer for a piece of data/info

Recall data types...

```
int num = 5; // declare a new integer var

char letter = 'x'; // b is a char ("character")

double decimal = 1.06; // d is a double, a type
used to represent decimal numbers

string sentence = "this is a C++ string";

bool isRaining = false;
```



This is why functions have return types (what type of info they output) and variables have types (what type of info they store)!

Recall data types...

```
int num = 5; // declare a new integer var

char letter = 'x'; // b is a char ("character")

double decimal = 1.06; // d is a double, a type
used to represent decimal numbers

string sentence = "this is a C++ string";

bool isRaining = false;
```

string
"this is a C++ string"
s

What's special about **strings** in C++?

Definition

string

A data type that represents a sequence of characters

Characters can be letters, digits, symbols (&, !, ~), etc.

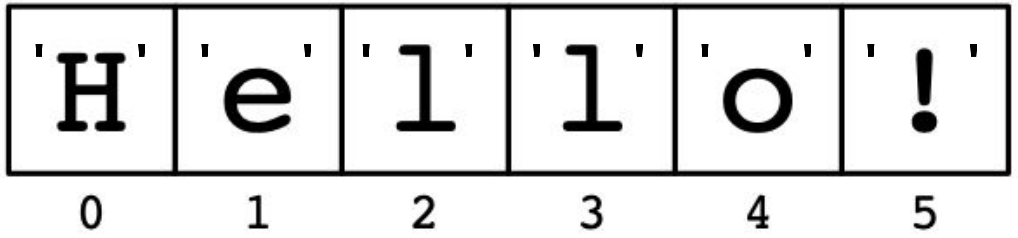
Definition

string

A data type that represents a sequence of characters

Strings review

Strings are made up of characters of type char, and the characters of a string can be accessed by the index in the string (this should be familiar):



string activity

[demo + poll]

What are the key characteristics of strings in C++?

- Strings are mutable in C++
  - Unlike in Python and Java
  - But you must assign string indices to a character:

YES: `word[1] = ‘a’;`

NO: `word[1] = “a”;`

What are the key characteristics of strings in C++?

- Strings are mutable in C++

- You can add characters to strings and strings to strings using += and +
    - Strings must use double quotes (“”) while characters use single (‘’).
    - There is a caveat you’ll see shortly

What are the key characteristics of strings in C++?

- Strings are mutable in C++

- You can add characters to strings and strings to strings using += and +
  - Strings must use double quotes (“”) while characters use single (‘’).
  - There is a caveat you’ll see shortly

- You can use logical operators to compare strings (and characters)
  - “Under-the-hood,” C++ is using the ASCII values of their first characters to compare.

string and char conventions

[demo]

string utility functions

Three categories of functions

- Built-in C++ char functions (<cctype> library)
- Built-in C++ string methods (<string> library)
- Stanford string library functions (“strlib.h” library)

<cctype> library

- **`#include <cctype>`**
- This library provides functions that check a single **`char`** for a property (e..g, if it is a digit), or return a **`char`** converted in some way (e.g., to uppercase)
  - **`isalnum`**: checks if a character is alphanumeric
  - **`isalpha`**: checks if a character is alphabetic
  - **`islower`**: checks if a character is lowercase
  - **`isupper`**: checks if a character is an uppercase character
  - **`isdigit`**: checks if a character is a digit
  - **`isxdigit`**: checks if a character is a hexadecimal character
  - **`iscntrl`**: checks if a character is a control character
  - **`isgraph`**: checks if a character is a graphical character
  - **`isspace`**: checks if a character is a space character
  - **`isblank`**: checks if a character is a blank character
  - **`isprint`**: checks if a character is a printing character
  - **`ispunct`**: checks if a character is a punctuation character
  - **`tolower`**: converts a character to lowercase
  - **`toupper`**: converts a character to uppercase

<cctype> library

- **`#include <cctype>`**
- This library provides functions that check a single **`char`** for a property (e..g, if it is a digit), or return a **`char`** converted in some way (e.g., to uppercase)
    - **`isalnum`**: checks if a character is alphanumeric
    - **`isalpha`**: checks if a character is alphabetic
    - **`islower`**: checks if a character is lowercase
    - **`isupper`**: checks if a character is an uppercase character
    - **`isdigit`**: checks if a character is a digit
    - **`isxdigit`**: checks if a character is a hexadecimal character
    - **`iscntrl`**: checks if a character is a control character
    - **`isgraph`**: checks if a character is a graphical character
    - **`isspace`**: checks if a character is a space character
    - **`isblank`**: checks if a character is a blank character
    - **`isprint`**: checks if a character is a printing character
    - **`ispunct`**: checks if a character is a punctuation character
    - **`tolower`**: converts a character to lowercase
    - **`toupper`**: converts a character to uppercase

```
char letter = 'L';
islower(letter);
//returns false
```

<cctype> library

- `#include <cctype>`
- This library provides functions that check a single **`char`** for a property (e..g, if it is a digit), or return a **`char`** converted in some way (e.g., to uppercase)
  - **`isalnum`**: checks if a character is alphanumeric
  - **`isalpha`**: checks if a character is alphabetic
  - **`islower`**: checks if a character is lowercase
  - **`isupper`**: checks if a character is an uppercase character
  - **`isdigit`**: checks if a character is a digit
  - **`isxdigit`**: checks if a character is a hexadecimal character
  - **`iscntrl`**: checks if a character is a control character
  - **`isgraph`**: checks if a character is a graphical character
  - **`isspace`**: checks if a character is a space character
  - **`isblank`**: checks if a character is a blank character
  - **`isprint`**: checks if a character is a printing character
  - **`ispunct`**: checks if a character is a punctuation character
  - **`tolower`**: converts a character to lowercase
  - **`toupper`**: converts a character to uppercase

string methods

`#include <string>`

- **`s.append(str)`**: add text **`str`** to the end of a string **`s`**
- **`s.compare(str)`**: return **`-1`**, **`0`**, or **`1`** depending on relative ordering
- **`s.erase(index, length)`**: delete text from a string starting at given **`index`**
- **`s.find(str)`**: return first index where the start of **`str`** appears in this string (returns **`string::npos`** if not found)
- **`s.rfind(str)`**: return last index where the start of **`str`** appears in this string (returns **`string::npos`** if not found)
- **`s.insert(index, str)`**: add text **`str`** into a string at a given **`index`**
- **`s.length()`** or **`s.size()`**: number of characters in this string
- **`s.replace(index, len, str)`**: replaces **`len`** chars at **`index`** with text **`str`**
- **`s.substr(start, length)`** or **`s.substr(start)`**: the next **`length`** characters beginning at **`start`** (inclusive); if **`length`** omitted, grabs until end of string

string methods

`#include <string>`

- **`s.append(str)`**: add text **`str`** to the end of a string **`s`**
- **`s.compare(str)`**: return **`-1`**, **`0`**, or **`1`** depending on relative ordering
- **`s.erase(index, length)`**: delete text from a string starting at given **`index`**
- **`s.find(str)`**: return first index where the start of **`str`** appears in this string (returns **`string::npos`** if not found)
- **`s.rfind(str)`**: return last index where the start of **`str`** appears in this string (returns **`string::npos`** if not found)
- **`s.insert(index, str)`**: add text **`str`** into a string at a given **`index`**
- **`s.length()`** or **`s.size()`**: number of characters in this string
- **`s.replace(index, len, str)`**: replaces **`len`** chars at **`index`** with text **`str`**
- **`s.substr(start, length)`** or **`s.substr(start)`**: the next **`length`** characters beginning at **`start`** (inclusive); if **`length`** omitted, grabs until end of string

Stanford string library functions

`#include “strlib.h”`

- **`endsWith(str, suffix)`**
  `startsWith(str, prefix)`: returns **`true`** if the given string begins or ends with the given **`suffix`**/**`prefix`** text
- **`integerToString(int)`**
  `realToString(double)`
  `stringToInteger(str)`
  `stringToReal(str)`: returns a conversion between numbers and strings
- **`equalsIgnoreCase(s1, s2)`**: **`true`** if **`s1`** and **`s2`** have same **`chars`**, ignoring casing
- **`toLowerCase(str)`**: returns a lowercase version of a string
- **`toUpperCase(str)`**: returns an uppercase version of a string
- **`trim(str)`**: returns string with surrounding whitespace removed

Stanford string library functions

`#include “strlib.h”`

- **`endsWith(str, suffix)`**
  `startsWith(str, prefix)`: returns **`true`** if the given string begins or ends with the given **`suffix`**/**`prefix`** text
- **`integerToString(int)`**
  `realToString(double)`
  `stringToInteger(str)`
  `stringToReal(str)`: returns a conversion between numbers and strings
- **`equalsIgnoreCase(s1, s2)`**: **`true`** if **`s1`** and **`s2`** have same **`chars`**, ignoring casing
- **`toLowerCase(str)`**: returns a lowercase version of a string
- **`toUpperCase(str)`**: returns an uppercase version of a string
- **`trim(str)`**: returns string with surrounding whitespace removed

Two types of C++ strings

[poll]

Poll: What will happen with the following line of code?

```
string hiThere = "hi" + "there";
```

You would get…

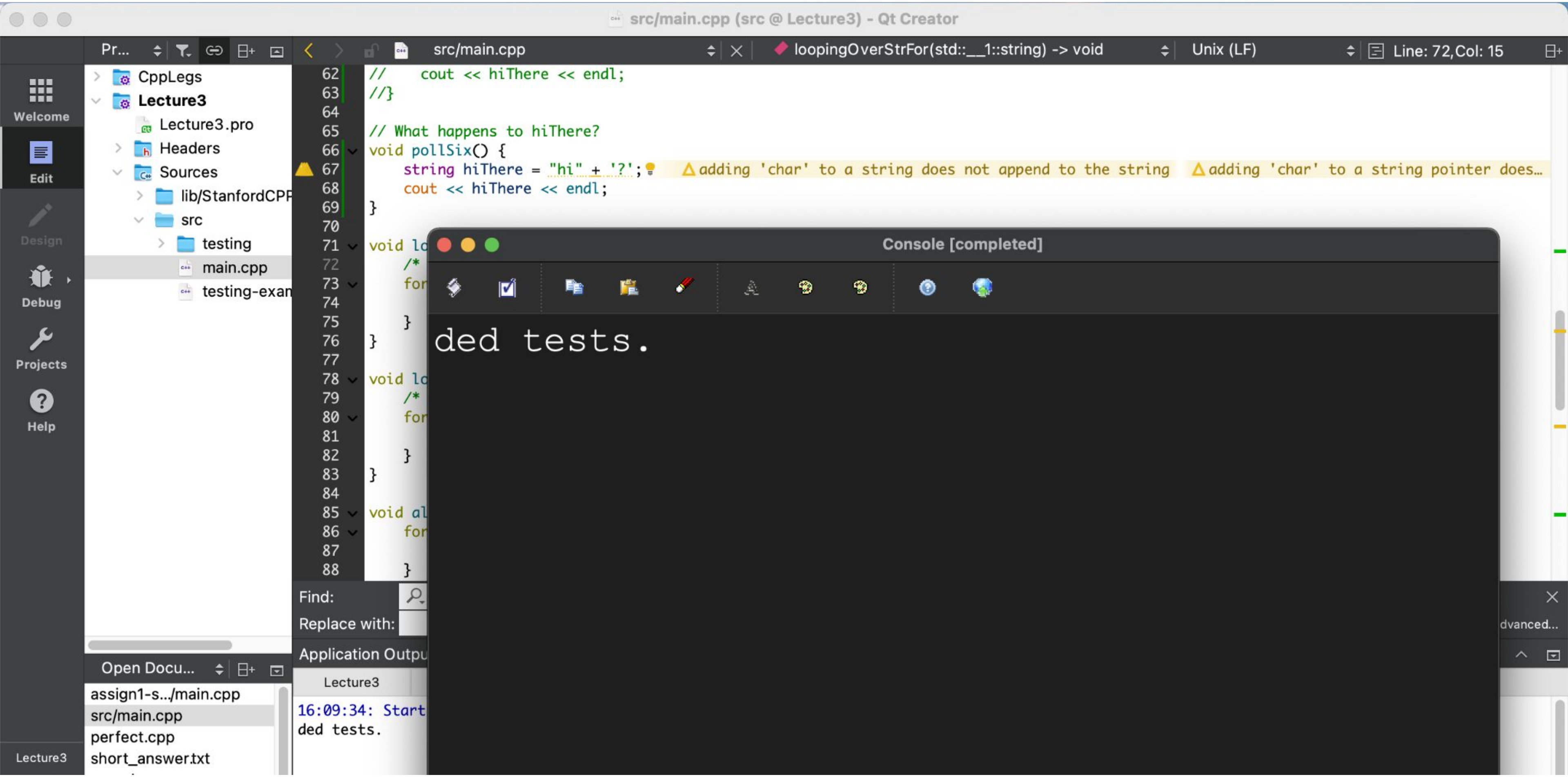
- An error
- The string "hithere" stored in `hiThere`
- The sum of the ASCII values of the letters in "hi" and "there"

Poll: What will happen with the following line of code?

```
string hiThere = "hi" + '?'
```

You would get…

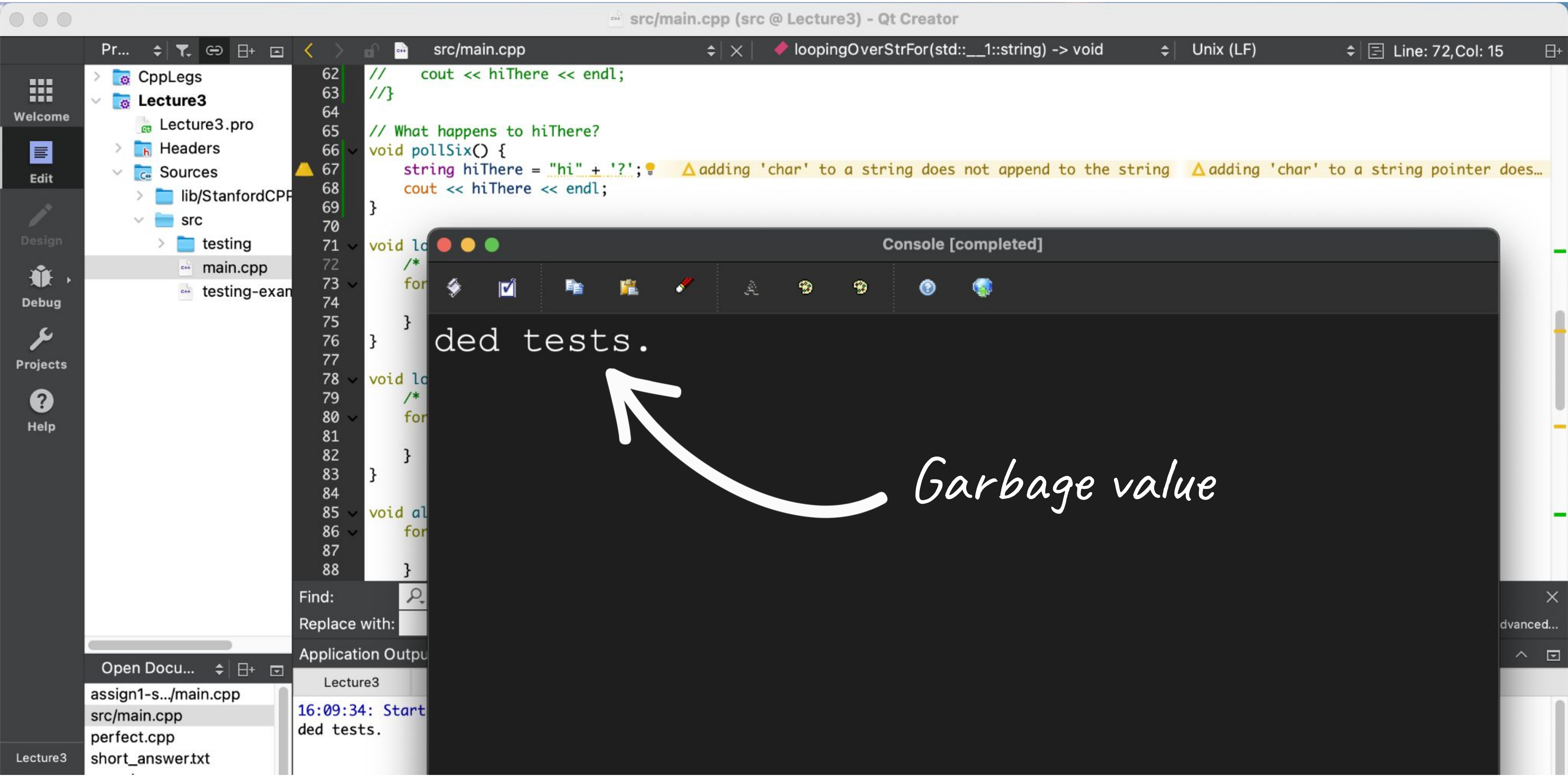
- An error
- The string “hi?” stored in `hiThere`
- “ded tests.”



```
//    cout << hiThere << endl;
//}

// What happens to hiThere?
void pollSix() {
    string hiThere = "hi" + '?';  ⚠adding 'char' to a string does not append to the string  ⚠adding 'char' to a string pointer does…
    cout << hiThere << endl;
}

void lo
    /*
    for

    }
}

void lo
    /*
    for

    }
}

void al
    for

    }
```

```
ded tests.
```

```
16:09:34: Start
ded tests.
```



```
//    cout << hiThere << endl;
//}

// What happens to hiThere?
void pollSix() {
    string hiThere = "hi" + '?';
    cout << hiThere << endl;
}

void lo
    /*
    for

    }
}

void lo
    /*
    for

    }
}

void al
    for

    }
```

⚠adding 'char' to a string does not append to the string ⚠adding 'char' to a string pointer does…

Console [completed]

```
ded tests.
```

Garbage value

```
16:09:34: Start
ded tests.
```

C strings vs. C++ strings summary

- C strings have no methods
    - This is why you can't do something like `"hi".length()` in C++

- Conversion fixes
    - **Store the C string in a variable first** to convert it to a C++ string
    - Use a conversion function
        - `string("text");` converts the C string literal into a C++ string
        - `string.c_str()` returns a C string from a C++ string

- **Takeaway**: Beware the C string!

Announcements

Announcements

- Sections signups are open at cs198.stanford.edu! Make sure to submit preferences by Sunday at 5pm PDT.

- Assignment 0 is due today at midnight.

- Assignment 1 will be released later today and is due next Friday, July 2 at 11:59pm PDT.
    - YEAH hours are after lecture next Monday, June 28 at 12:30pm PDT.

- Please send us your OAE letters as soon as possible if you haven't already.

How do we **test** code in CS106B?

Testing

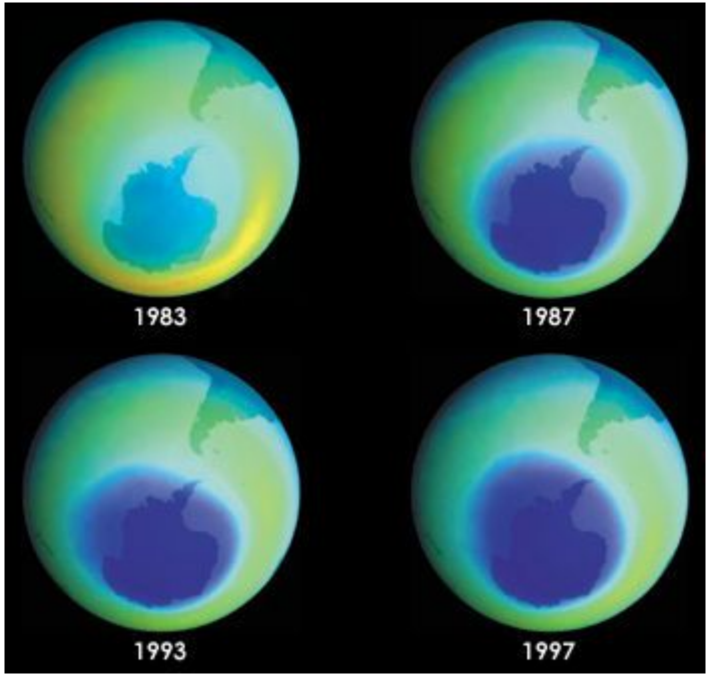
Software and cathedrals are much the same – first we build them,
then we pray.
– Sam Redwine

Why is testing important?

Discuss in breakout rooms!

Why is testing important?

The hole in the ozone layer over Antarctica remained undetected for a long period of time because the data analysis software used by NASA in its project to map the ozone layer had been **designed to ignore values that deviated greatly from expected measurements.**



Source

Why is testing important?

In 1996, a European Ariane 5 rocket was set to deliver a payload of satellites into Earth orbit, but problems with the software caused the launch rocket to veer off its path a mere 37 seconds after launch. The problem was the result of **code reuse from the launch system's predecessor, Ariane 4, which had very different flight conditions from Ariane 5.**

Source

Why is testing important?

A 2002 study commissioned by the National Institute of Standards and Technology (referred to here) **found that software bugs cost the U.S. economy $59.5 billion every year** (imagine the global costs…). The study estimated that more than a third of that amount, $22.2 billion, **could be eliminated by improved testing.**

Source

Why is testing important?

- Testing can save money

Why is testing important?

- Testing can save money

- Testing can save lives

Why is testing important?

- Testing can save money

- Testing can save lives

- Testing can prevent disasters

Why is testing important?

- Testing can save money
- Testing can save lives
- Testing can prevent disasters
- **Testing is a programmer's responsibility.**
  - You must think about ethical considerations when you develop code that impacts people.

What are good testing strategies?

What are good testing strategies?

- Write tests that cover a wide variety of use cases for your function!

What are good testing strategies?

- Write tests that cover a wide variety of use cases for your function!
    - Use your critical thinking and analysis skills to identify a diverse range of possible ways in which your code might be used.

What are good testing strategies?

- Write tests that cover a wide variety of use cases for your function!

- Consider:
    - Basic use cases
    - Edge cases

What are good testing strategies?

- Write tests that cover a wide variety of use cases for your function!

- Consider:
  - Basic use cases
  - Edge cases

Definition

edge case

Uses of your function/program that represent extreme situations

What are good testing strategies?

- Write tests that cover a wide variety of use cases for your function!
- Consider:
  - Basic use cases
  - Edge cases

For example, if your function takes in an integer parameter, test what happens if the value that is passed in negative, zero, a large positive number, etc!

Definition

edge case

Uses of your function/program that represent extreme situations

SimpleTest

What is SimpleTest?

- SimpleTest is a C++ library developed by some of the lecturers here at Stanford that allows standalone, C++ unit testing

- For those of you coming from CS106A in Python, this is similar in functionality to the **doctest** infrastructure that you learned

- We will see SimpleTest a lot this quarter! You will learn how to write good, comprehensive suites of tests using this library, starting from the very first assignment.

How does SimpleTest work?

CS106B Testing Guide
– make sure to read it!

How does SimpleTest work?

main.cpp

```
#include "testing/SimpleTest.h"
#include "testing-examples.h"

int main()
{
    if (runSimpleTests(SELECTED_TESTS)) {
        return 0;
    }

    return 0;
}
```

```
NO_TESTS
SELECTED_TESTS
ALL_TESTS
```

How does SimpleTest work?

main.cpp

```
#include "testing/SimpleTest.h"
#include "testing-examples.h"

int main()
{
    if (runSimpleTests(SELECTED_TESTS)) {
        return 0;
    }

    return 0;
}
```

testing-examples.cpp

```
#include "testing/SimpleTest.h"

int factorial (int num);

int factorial (int num) {
     /* Implementation here */
}

PROVIDED_TEST("Some provided tests.") {
     EXPECT_EQUAL(factorial(1), 1);
     EXPECT_EQUAL(factorial(2), 2);
     EXPECT_EQUAL(factorial(3), 6);
     EXPECT_EQUAL(factorial(4), 24);
}

STUDENT_TEST("student wrote this test") {
     // student tests go here!
}
```

How does SimpleTest work?

main.cpp

```
#include "testing/SimpleTest.h"
#include "testing-examples.h"

int main()
{
    if (runSimpleTests(SELECTED_TESTS)) {
        return 0;
    }

    return 0;
}
```

testing-examples.cpp

```
#include "testing/SimpleTest.h"

int factorial (int num);

int factorial (int num) {
     /* Implementation here */
}

PROVIDED_TEST("Some provided tests.") {
     EXPECT_EQUAL(factorial(1), 1);
     EXPECT_EQUAL(factorial(2), 2);
     EXPECT_EQUAL(factorial(3), 6);
     EXPECT_EQUAL(factorial(4), 24);
}

STUDENT_TEST("student wrote this test") {
     // student tests go here!
}
```

How does SimpleTest work?

<u>main.cpp</u>

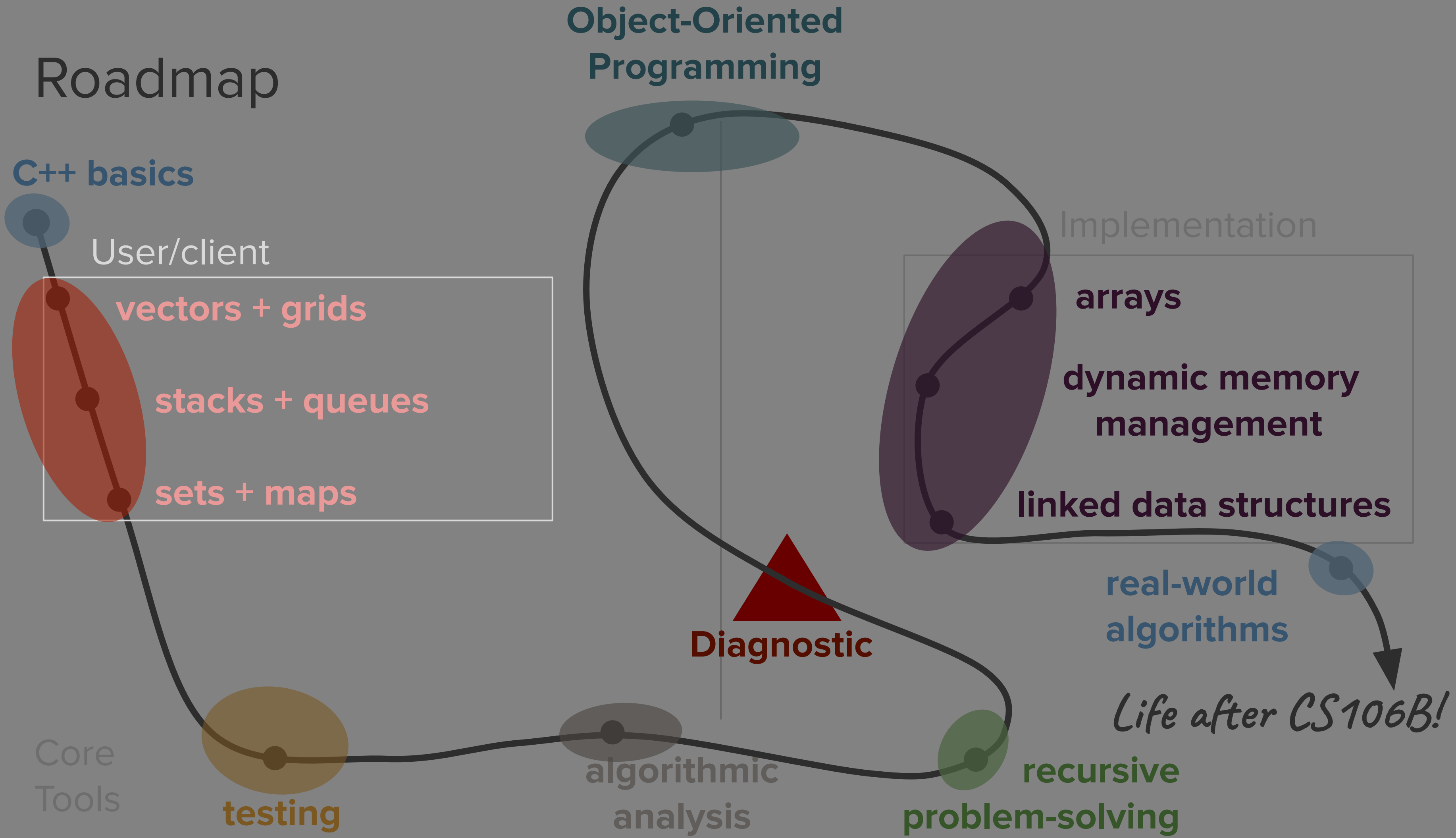
```
#include "testing/SimpleT
#include "testing-examp

int main()
{
    if (runSimpleTests(
        return 0;
    }

    return 0;
}
```

<u>testing-examples.cpp</u>

```
                   ng/SimpleTest.h"

                   int num);

                   int num) {
                   ntation here */



                   Some provided tests.") {
                   AL(factorial(1), 1);
                   AL(factorial(2), 2);
                   AL(factorial(3), 6);
                 QUAL(factorial(4), 24);



STUDENT_TEST("student wrote this test") {
      // student tests go here!
}
```

What’s next?

Roadmap


Object-Oriented Programming
C++ basics
User/client
vectors + grids
stacks + queues
sets + maps
Implementation
arrays
dynamic memory management
linked data structures
real-world algorithms
Diagnostic
Life after CS106B!
Core Tools
testing
algorithmic analysis
recursive problem-solving

Vectors and Grids