Programming Fundamentals in C++

What programming language are you most comfortable with?

(put your answers the chat)
Roadmap

C++ basics

User/client

vectors + grids
stacks + queues
sets + maps

Object-Oriented Programming

arrays
dynamic memory management
linked data structures

Core Tools

testing
algorithmic analysis

Implementation

Diagnostic

real-world algorithms
recursive
test

Life after CS106B!
Today’s questions

Why C++?

What do core programming fundamentals look like in C++?

What’s next?
Why C++?
How is C++ different from other languages?

- C++ is a compiled language (vs. interpreted)
  - This means that before running a C++ program, you must first compile it to machine code.
How is C++ different from other languages?

- C++ is a compiled language (vs. interpreted)

- C++ gives us access to lower-level computing resources (e.g. more direct control over computer memory)
  - This makes it a great tool for better understanding abstractions!
How is C++ different from other languages?

- C++ is a compiled language (vs. interpreted)

- C++ gives us access to lower-level computing resources (e.g. more direct control over computer memory)

- If you’re coming from a language like Python, the syntax will take some getting used to.
  - Like learning the grammar and rules of a new language, typos are expected. But don’t let this get in the way of working toward literacy!
How is C++ different from other languages?

- C++ is a compiled language (vs. interpreted)
- C++ gives us access to lower-level computing resources (e.g. more direct control over computer memory)
- If you’re coming from a language like Python, the syntax will take some getting used to.
Demo program!
The structure of a program

```cpp
#include <iostream>
#include "console.h"
using namespace std;

// The C++ compiler will look for a function
// called “main”
int main() {
    cout << "Hello, world!" << endl;
    return 0; // must return an int to indicate
               // successful program completion
}
```

```python
import sys

# This function does not need to be called “main”
def main():
    print('Hello, world!')

if __name__ == '__main__':
    # Any function that gets placed here will get
    # called when you run the program with
    # `python3 helloworld.py`
    main()
```

C++Python
Zoom Poll!

Where does C++ rank among the popular programming languages of the world?
C++ Overview

If someone claims to have the perfect programming language, he is either a fool or a salesman or both.
– Bjarne Stroustrup, Inventor of C++
C++ History

- C++ is a high-performance, robust (and complex) language built on top of the C programming language (originally named C with Classes)
  - Bjarne Stroustrup, the inventor of C++, chose to build on top of C because it was fast, powerful, and widely-used
C++ History

- C++ is a high-performance, robust (and complex) language built on top of the C programming language (originally named C with Classes)
  - Bjarne Stroustrup, the inventor of C++, chose to build on top of C because it was fast, powerful, and widely-used

- C++ has been an object-oriented language from the beginning
  - We will spend the middle portion of this class talking about the paradigm of object-oriented programming
C++ History

- C++ is a high-performance, robust (and complex) language built on top of the C programming language (originally named C with Classes)
  - Bjarne Stroustrup, the inventor of C++, chose to build on top of C because it was fast, powerful, and widely-used
- C++ has been an object-oriented language from the beginning
  - We will spend the middle portion of this class talking about the paradigm of object-oriented programming
- C++ is quite mature and has become complex enough that it is challenging to master the language
  - Our goal in this class will be to help you become literate in C++ as a second programming language
C++ History

- C++ is a high-performance, robust (and complex) language built on top of the C programming language (originally named C with Classes)
  - Bjarne Stroustrup, the inventor of C++, chose to build on top of C because it was fast, powerful, and widely-used
- C++ has been an object-oriented language from the beginning
  - We will spend the middle portion of this class talking about the paradigm of object-oriented programming
- C++ is quite mature and has become complex enough that it is challenging to master the language
  - Our goal in this class will be to help you become literate in C++ as a second programming language
C++ Benefits and Drawbacks

Benefits
C++ Benefits and Drawbacks

Benefits

- C++ is **fast**
  - Get ready for the Python vs C++ speed showdown during Assignment 1!
C++ Benefits and Drawbacks

Benefits

- C++ is **fast**
  - Get ready for the Python vs C++ speed showdown during Assignment 1!
- C++ is **popular**
  - Many companies and research projects use C++ and it is common for coding interviews to be conducted in C++
C++ Benefits and Drawbacks

**Benefits**

- **C++ is fast**
  - Get ready for the Python vs C++ speed showdown during Assignment 1!

- **C++ is popular**
  - Many companies and research projects use C++ and it is common for coding interviews to be conducted in C++

- **C++ is powerful**
  - C++ brings you closer to the raw computing power that your computer has to offer
## C++ Benefits and Drawbacks

<table>
<thead>
<tr>
<th>Benefits</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C++ is</strong> fast</td>
<td></td>
</tr>
<tr>
<td>○ Get ready for the Python vs C++ speed showdown during Assignment 1!</td>
<td></td>
</tr>
<tr>
<td><strong>C++ is</strong> popular</td>
<td></td>
</tr>
<tr>
<td>○ Many companies and research projects use C++ and it is common for coding interviews to be conducted in C++</td>
<td></td>
</tr>
<tr>
<td><strong>C++ is</strong> powerful</td>
<td></td>
</tr>
<tr>
<td>○ C++ brings you closer to the raw computing power that your computer has to offer</td>
<td></td>
</tr>
</tbody>
</table>
C++ Benefits and Drawbacks

**Benefits**

- **C++ is fast**
  - Get ready for the Python vs C++ speed showdown during Assignment 1!

- **C++ is popular**
  - Many companies and research projects use C++ and it is common for coding interviews to be conducted in C++

- **C++ is powerful**
  - C++ brings you closer to the raw computing power that your computer has to offer

**Drawbacks**

- **C++ is complex**
  - We will rely on the Stanford C++ libraries to provide a friendlier level of abstraction
  - In the future, you may choose to explore the *standard* libraries
C++ Benefits and Drawbacks

**Benefits**

- **C++ is fast**
  - Get ready for the Python vs C++ speed showdown during Assignment 1!
- **C++ is popular**
  - Many companies and research projects use C++ and it is common for coding interviews to be conducted in C++
- **C++ is powerful**
  - C++ brings you closer to the raw computing power that your computer has to offer

**Drawbacks**

- **C++ is complex**
  - We will rely on the Stanford C++ libraries to provide a friendlier level of abstraction
  - In the future, you may choose to explore the standard libraries
- **C++ can be dangerous**
  - "With great power comes great responsibility"
What do core programming fundamentals look like in C++?
What do core programming fundamentals look like in C++?

Get ready for a whirlwind tour!
Comments, Includes, and Console Output
Comments

- Single-line comments

```cpp
// Two forward slashes comment out the rest of the line

cout << "Hello, World!" << endl; // everything past the double-slash is a comment
```

- Multi-line comments

```cpp
/* This is a multi-line comment.

* It begins and ends with an asterisk-slash.

*/
```
Includes

- Utilizing code written by other programmers is one of the most powerful things that you can do when writing code.
- In order to make the compiler aware of other code libraries or other code files that you want to use, you must **include a header file**. There are two ways that you can do so:
  - `#include <iostream>`
    - Use of the angle bracket operators is usually reserved for code from the C++ Standard library
  - `#include "console.h"
    - Use of the quotes is usually reserved for code from the Stanford C++ libraries, or code in files that you have written yourself
The console is the main venue that we will use in this class to communicate information from a program to the user of the program.
The console is the main venue that we will use in this class to communicate information from a program to the user of the program.

In C++, the way that you get information to the console is by using the `cout` keyword and angle bracket operators (`<<`).

```cpp
cout << "The answer to life, the universe, and everything is " << 42 << "." << endl;
```

The answer to life, the universe, and everything is 42.
The console is the main venue that we will use in this class to communicate information from a program to the user of the program.

In C++, the way that you get information to the console is by using the `cout` keyword and angle bracket operators (`<<`).

The `endl` is necessary to put the cursor on a different line. Here is an example with and without the `endl` keyword.

```cpp
cout << "This is some text followed by endl." << endl;
cout << "This is more text.";
cout << "We want to go to the next line here, too" << endl;
cout << "We made it to the next line." << endl;
```
The console is the main venue that we will use in this class to communicate information from a program to the user of the program. In C++, the way that you get information to the console is by using the `cout` keyword and angle bracket operators (`<<`).

The `endl` is necessary to put the cursor on a different line. Here is an example with and without the `endl` keyword.

```cpp
cout << "This is some text followed by endl." << endl;
cout << "This is more text.";
cout << "We want to go to the next line here, too" << endl;
cout << "We made it to the next line." << endl;
```

Note: In C++, all programming statements must end in a semicolon.
Variables and Types
Variables

- A way for code to store information by associating a value with a name
Variables

- A way for code to store information by associating a value with a name

- `classNum`: 106
- `tuesdayTemp`: 94.7
Variables

- A way for code to store information by associating a value with a name.

We will think of a variable as a named container storing a value.

```
106  classNum
94.7  tuesdayTemp
```
Variables

- A way for code to store information by associating a value with a name

Note: C++ uses the camelCase naming convention

- classNum: 106
- tuesdayTemp: 94.7
Variables

- A way for code to store information by associating a value with a name
- Variables are perhaps one of the most fundamental aspects of programming! Without variables, the expressive power of our computer programs would be severely degraded.
As you should know from prior programming classes, all variables have a type associated with them, where the type describes the representation of the variable.
Types

- As you should know from prior programming classes, all variables have a type associated with them, where the type describes the representation of the variable.
- Examples of types in C++
  - `int` (or `long`)
Types

- As you should know from prior programming classes, all variables have a type associated with them, where the type describes the representation of the variable.
- Examples of types in C++
  - `int` (or `long`)
  - `double`
Types

- As you should know from prior programming classes, all variables have a type associated with them, where the type describes the representation of the variable.

- Examples of types in C++
  - int (or long)
  - double
  - string

"Hello, World!"
"CS106B"

"I love computer science <3"
Types

- As you should know from prior programming classes, all variables have a type associated with them, where the type describes the representation of the variable.
- Examples of types in C++
  - `int` (or `long`)
  - `double`
  - `string`
  - `char`
Types

- As you should know from prior programming classes, all variables have a type associated with them, where the type describes the representation of the variable.
- Examples of types in C++
  - `int` (or `long`)
  - `double`
  - `string`
  - `char`
- In C++, **all types must be explicitly defined when the variable is created, and a variable cannot change its type.**
Typed Variables

```c
int a; // declare a new integer variable
```
Typed Variables

```c
int a; // declare a new integer variable
a = 5; // initialize the variable value
```
Typed Variables

```c
int a; // declare a new integer variable
a = 5; // initialize the variable value
char b = 'x'; // b is a char
("character")
```
Typed Variables

```c
int a; // declare a new integer variable
a = 5; // initialize the variable value
char c = 'x'; // b is a char ("character")
double d = 1.06; // d is a double, a type used to represent decimal numbers
```
Typed Variables

```cpp
int a; // declare a new integer variable
a = 5;  // initialize the variable value
char c = 'x'; // b is a char ("character")
double d = 1.06; // d is a double, a type used to represent decimal numbers
string s = "this is a C++ string";
```
Typed Variables

```cpp
int a;  // declare a new integer variable
a = 5;  // initialize the variable value
char c = 'x';  // b is a char ("character")
double d = 1.06;  // d is a double, a type used to represent decimal numbers
string s = "this is a C++ string";
double a = 4.2;  // ERROR! You cannot redefine a variable to be another type
```
Typed Variables

```cpp
int a; // declare a new integer variable
a = 5; // initialize the variable value
char c = 'x'; // b is a char ("character")
double d = 1.06; // d is a double, a type used to represent decimal numbers
string s = "this is a C++ string";
double a = 4.2; // ERROR! You cannot redefine a variable to be another type
int a = 12; // ERROR! You do not need the type when re-assigning a variable
```
Typed Variables

```cpp
int a; // declare a new integer variable
a = 5; // initialize the variable value
char c = 'x'; // b is a char ("character")
double d = 1.06; // d is a double, a type used to represent decimal numbers
string s = "this is a C++ string";
double a = 4.2; // ERROR! You cannot redefine a variable to be another type
int a = 12; // ERROR! You do not need the type when re-assigning a variable
a = 12; // this is okay, updates variable value
```
Mid-Lecture
Announcements
Break!
Announcements

● Complete the [C++ survey](#) and help us plan tomorrow's lecture!
● Fill out your section time preferences by Sunday at 5pm PDT (opens tomorrow at 5pm PDT).
● Finish [Assignment 0](#) by Friday at 11:59 pm PDT.
  ○ If you’re running into issues with Qt Creator, come to the Qt Installation Help Session Thursday at 5pm PDT. Join at the LaIR link ([cs198.stanford.edu/lair](http://cs198.stanford.edu/lair)) to get help.
● Assignment 1 will be released Friday, and after this lecture is over, you will have the skills you need to get started on the first part!
  ○ There be a YEAH (Your Early Assignment Help) session held from 12:30-1:30pm PDT on Monday afternoon to help folks get started.
Functions and Parameters
Anatomy of a function

input -> function(input) -> output
Anatomy of a function

input → function(input) → output

parameters/arguments
Anatomy of a function

**Definition**

**parameter(s)**

One or more variables that your function expects as input
Anatomy of a function

**Definition**

**argument(s)**

The values passed into your function and assigned to its parameter variables.
Anatomy of a function

input → function(input) → output

return value
Anatomy of a function

Definition

**return value**
The value that your function hands back to the “calling” function.
Anatomy of a function

input → function(input) → output

parameters/arguments → return value
Anatomy of a function

```
returnType functionName(varType parameter1, varType parameter2, ...);
```
Anatomy of a function

`returnType functionName(varType parameter1, varType parameter2, ...);`
Anatomy of a function

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

input expected
(parameters)
Anatomy of a function

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

Notice that these look very similar to variable declarations!
You can think of parameters as a special set of local variables that belong to a function.
Anatomy of a function

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

output expected
(return type)
Anatomy of a function

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

How do you designate a function that doesn’t return a value? You can use the special `void` keyword. Note that this type is only applicable for return types, not parameters/variables.
Anatomy of a function

```c
returnType functionName(varType parameter1, varType parameter2, ...);
```
Anatomy of a function

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

returnType functionName(varType parameter1, varType parameter2, ...) {
    returnType variable = /* Some fancy code. */
    /* Some more code to actually do things. */
    return variable;
}
```
Function Example

double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}

int main() {
    double mid = average(10.6, 7.2);
    cout << mid << endl;
    return 0;
}
Function Example

double average(double a, double b){
    double sum = a + b;
    return sum / 2;
}

int main(){
    double mid = average(10.6, 7.2);
    cout << mid << endl;
    return 0;
}
double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}

int main() {
    double mid = average(10.6, 7.2);
    cout << mid << endl;
    return 0;
}
Function Example

double average(double a, double b){
    double sum = a + b;
    return sum / 2;
}

int main(){
    double mid = average(10.6, 7.2);
    cout << mid << endl;
    return 0;
}
Function Example

double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}

int main() {
    double mid = average(10.6, 7.2);
    cout << mid << endl;
    return 0;
}
Function Example

double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}

int main() {
    double mid = average(10.6, 7.2);
    cout << mid << endl;
    return 0;
}
Function Example

double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}

int main() {
    double mid = average(10.6, 7.2);
    cout << mid << endl;
    return 0;
}
double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}

int main() {
    double mid = average(10.6, 7.2);
    cout << mid << endl;
    return 0;
}
Pass by Value

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

int doubleValue(int x) {
    x *= 2;
    return x;
}

int main() {
    int myValue = 5;
    int result = doubleValue(myValue);

    cout << "myValue: " << myValue << " ";
    cout << "result: " << result << endl;
}
```

Zoom Poll!

What is the console output of this block of code?
Pass by Value

// C++:
#include<iostream>
using namespace std;

int doubleValue(int x) {
    x *= 2;
    return x;
}

int main() {
    int myValue = 5;
    int result = doubleValue(myValue);

    cout << "myValue: " << myValue << " " << result << endl;
}

myValue: 5 result: 10

Why is this the case?
Pass by Value

// C++:
#include <iostream>
using namespace std;

int doubleValue(int x) {
  x *= 2;
  return x;
}

int main() {
  int myValue = 5;
  int result = doubleValue(myValue);

  cout << "myValue: " << myValue << " ";
  cout << "result: " << result << endl;
}

- The reason for the output is that the parameter x was passed to the `doubleValue` function by value, meaning that the variable x is a copy of the variable passed in. Changing it inside the function does not change the value in the calling function.
- Pass-by-value is the default mode of operation when it comes to parameters in C++
- C++ also supports a different, more nuanced way of passing parameters – we will see this in the next lecture!
Control Flow
### Boolean Expressions

<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a &lt; b</td>
<td>a is less than b</td>
<td>a &amp;&amp; b</td>
<td>Both a AND b are true</td>
</tr>
<tr>
<td>a &lt;= b</td>
<td>a is less than or equal to b</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>a &gt; b</td>
<td>a is greater than b</td>
<td>!a</td>
<td>If a is true, returns false, and vice-versa</td>
</tr>
<tr>
<td>a &gt;= b</td>
<td>a is greater than or equal to b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a == b</td>
<td>a is equal to b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a != b</td>
<td>a is not equal to b</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conditional Statements

- The C++ if statement tests a boolean expression and runs a block of code if the expression is true, and, optionally, runs a different block of code if the expression is false. The if statement has the following format:

```
  if (expression) {
    statements if expression is true
  } else {
    statements if expression is false
  }
```

Note: The parentheses around expression are required.
Conditional Statements

● The C++ if statement tests a boolean expression and runs a block of code if the expression is **true**, and, optionally, runs a different block of code if the expression is **false**. The if statement has the following format:

  ○ if (expression) {
    statements if expression is true
  } else {
    statements if expression is false
  }

  **Note:** The parentheses around expression are **required**.

● In Python, a block is defined as an indentation level, where *whitespace* is important. C++ does not have any whitespace restrictions, so blocks are denoted with curly braces, { to begin a block, and } to end a block.

● Blocks are used primarily for conditional statements, functions, and loops.
Conditional Statements

- The C++ `if` statement tests a boolean expression and runs a block of code if the expression is `true`, and, optionally, runs a different block of code if the expression is `false`. The `if` statement has the following format:
  
  ```
  if (expression) {
    statements if expression is true
  }
  else {
    statements if expression is false
  }
  
  if (expression1) {
    statements if expression1 is true
  }
  else if (expression2) {
    statements if expression2 is true
  }
  else {
    statements if neither expression1 nor expression2 is true
  }
  ```

- Additional else if statements can be used to check for additional conditions as well.
**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

```c
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

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

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

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

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

Output:

0
1
2
3
4

Note: The `i++` increments the variable `i` by 1, and is the reason C++ got its name! (and there is a corresponding decrement operator, `--`, as in `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 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

```java
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

```java
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

```java
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

```java
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

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

```cpp
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

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

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

**Output:**

```
0
1
2
```
Interactive Example

[Breakout Room + Ed Workspaces]
Try it for yourself!

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 write “Liftoff.”
Try it for yourself!

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 write “Liftoff.”

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

if __name__ == "__main__":
    main()
```
Try it for yourself!

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 write “Liftoff.”

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

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

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

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

    return 0;
}
```
Try it for yourself!

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 write “Liftoff.”

Python

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

if __name__ == "__main__":
    main()

C++

#include <iostream>
using namespace std;

int main()
{
    /* TODO: Your code goes here! */
    return 0;
}

Breakout Rooms!
What’s next?
Strings, Testing, C++ Review