C++ Fundamentals

Elyse Cornwall
June 27th, 2023
Announcements and Reminders

• **Sign up for section** by today at 5pm!
  • Also, attend section this week
• Send OAE letters to Amrita and Elyse
• Assignment 0 due Friday at 11:59pm
• We’ll have our first attendance ticket in lecture today…
CS106B Roadmap

Using Abstractions

Object-Oriented Programming

Abstract Data Structures

Core Tools

C++

Algorithmic Analysis

Building Abstractions

Memory Management

Linked Data Structures

Advanced Algorithms

Recursion
CS106B Roadmap

Core Tools

C++

Using Abstractions

Abstract Data Structures

Object-Oriented Programming

Building Abstractions

Memory Management

Linked Data Structures

Advanced Algorithms

Algorithmic Analysis

Recursion
What programming languages have you used before?
Programming Language Popularity

TIOBE Programming Community Index
Source: www.tiobe.com
Programming Language Popularity

Stanford University
What is C++?

- High performance programming language, based on C
- Object-oriented language (we’ll explore this later in our roadmap)
  - “C with Classes”
- Huge! Complex!
Pros and Cons of C++

Pros

• C++ is fast
  • Between 10 and 100 times faster than Python!
• C++ is powerful
  • Allows more control over your computer’s resources
• C++ is popular
  • Coding interviews, research, industry
Pros and Cons of C++

Pros

• C++ is fast
  • Between 10 and 100 times faster than Python!
• C++ is powerful
  • Allows more control over your computer’s resources
• C++ is popular
  • Coding interviews, research, industry

Cons

• C++ is complex
  • We’ll be using some Stanford-specific libraries to make the interface friendlier (think abstraction)
• C++ can be dangerous
  • We can make memory errors and cause more severe crashes!
Let’s look at some C++ code!
Our First C++ Program

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

int main() {
    cout << "Hello, World!" << endl;
    return 0;
}
Our First C++ Program

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

int main() {
    cout << "Hello, World!" << endl;
    return 0;
}

Including libraries allows us to use code that was written elsewhere by somebody else
Our First C++ Program

#include "console.h"  \(\text{Directs user input / output to console}\)
#include <iostream>  \(\text{Standard input / output library}\)
using namespace std;

int main() {
    cout << "Hello, World!" << endl;
    return 0;
}

Our First C++ Program

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

int main() {
    cout << "Hello, World!" << endl;
    return 0;
}
```

Compiler looks for a function called main and starts program from there
Our First C++ Program

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

int main() {
    cout << "Hello, World!" << endl;
    return 0;
}

Function bodies are enclosed within “curly braces”
#include "console.h"
#include <iostream>
using namespace std;

int main() {
    cout << "Hello, World!" << endl;
    return 0;
}
Our First C++ Program

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

int main() {
    cout << "Hello, World!" << endl;
    return 0;
}
```

This is how we print to the console for the user to see.
Our First C++ Program

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

int main() {
    cout << "Hello, World!" << endl;
    return 0;
}
```

This is how we print to the console for the user to see (more on that later).
Brief Detour: Console Output

- We use `cout` and `<<` to print information to the user
- To start printing on a new line, we use `endl`
Brief Detour: Console Output

- We use `cout` and `<<` to print information to the user
- To start printing on a new line, we use `endl`

```cpp
int main()
{
    cout << "Hello, World!" << endl;
    cout << "Hello, World!" << endl;
    cout << "Hello, World!" << endl;
    return 0;
}
```
Brief Detour: Console Output

- We use `cout` and `<<` to print information to the user.
- To start printing on a new line, we use `endl`.

```cpp
int main()
{
    cout << "Hello, World!";
    cout << "Hello, World!";
    cout << "Hello, World!";
    return 0;
}
```
Brief Detour: Console Input

- We use `getLine()` with a prompt to get information from the user.
- `getLine()` returns a string, which we often store in a variable.

```cpp
int main()
{
    string name = getline("What's your name? ");
    cout << "Hello, " << name << endl;
    return 0;
}
```
Brief Detour: Console Programs

- In combination, cout and getline() let us communicate with the user via the console.
- Programs that do this are called “console programs”
Our First C++ Program

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

int main() {
    cout << "Hello, World!" << endl;
    return 0; // The main function returns 0 to indicate success
}
Variables and Types
Variables

- We use variables to store information in our programs
- Variables have a *type* and a *name*

```c
int enrollment;

string className;
```
Variables

- We use variables to store information in our programs
- Variables have a *type* and a *name*

```c
int enrollment;

string className;
```

*We name variables using “camelCase” capitalization*
Variable Types

- When we declare a variable, we must specify its type
- A variable cannot change type

```plaintext
int enrollment;  // create integer variable
enrollment = 190;  // set its value to 190
enrollment = 191;  // reassign its value to 191
```
Variable Types

• When we declare a variable, we must specify its type
• A variable cannot change type

```cpp
int enrollment;  // create integer variable
enrollment = 190;  // set its value to 190
enrollment = 191;  // reassign its value to 191
```

Before we set its value, this variable holds “garbage” data. It’s not initialized to 0 or cleared out for us.
Variable Types

• When we declare a variable, we must specify its type
• A variable cannot change type

```c
int enrollment; // create integer variable
enrollment = 190; // set its value to 190
enrollment++;    // reassign its value to 191
```
Variable Types

• When we declare a variable, we must specify its type
• A variable cannot change type

We only specify the type when first declaring the variable

```plaintext
int enrollment;       // create integer variable
enrollment = 190;     // set its value to 190
enrollment++;         // reassign its value to 191
```
Variable Types

• When we declare a variable, we must specify its type
• A variable cannot change type

```cpp
int enrollment;  // create integer variable
enrollment = 190;  // set its value to 190
enrollment = "full";  // ERROR!
```
C++ Types

Numbers

• int, long    // 100
• float, double    // 3.14

Text

• char, string    // ‘a’, “apple”

Booleans

• bool    // true, false
What is the value stored in the variable `mystery` after the following two lines of code execute?

```java
int mystery = 4;
mystery = 12;
```

Enter your answer on Gradescope by next class (SCPD students have until Sunday 11:59pm)
Functions, Parameters, and Returns
Functions

Parameters  Function  Return
Parameters and Returns

• Parameters: what information needs to be given to this function when it’s called?
• Return: what information should this function give back to whoever called it?
  • Often the result of a computation or the “final answer”
• Some functions don’t have parameters or returns
Example: function that sums two numbers

Parameters ➔ Function ➔ Return

Two numbers ➔ Their sum

Stanford University
Defining Functions in C++

• Choose a function name
  • We use camelCase just like variable names
• Define the name and type of any parameters
• Define the return type
  • Return type is `void` if the function doesn’t return anything

```cpp
int sum(int val1, int val2);
```
Defining Functions in C++

- Choose a function name
  - We use camelCase just like variable names
- Define the name and type of any parameters
- Define the return type
  - Return type is `void` if the function doesn’t return anything

```cpp
int sum(int val1, int val2);
```
Defining Functions in C++

- Choose a function name
  - We use camelCase just like variable names
- Define the **name and type of any parameters**
- Define the return type
  - Return type is `void` if the function doesn’t return anything

```cpp
int sum(int val1, int val2);
```
Defining Functions in C++

• Choose a function name
  • We use camelCase just like variable names
• Define the name and type of any parameters
• Define the return type
  • Return type is void if the function doesn’t return anything

```cpp
int sum(int val1, int val2);
```
Defining Functions in C++

• Choose a function name
  • We use camelCase just like variable names
• Define the name and type of any parameters
• Define the return type
  • Return type is `void` if the function doesn’t return anything

```cpp
int sum(int val1, int val2) {
    int result = val1 + val2;
    return result;
}
```
Function Order

• The order in which functions are defined matters in C++
• You cannot call a function before it’s been defined or declared

```cpp
int sum(int val1, int val2) {
    int result = val1 + val2;
    return result;
}

int main() {
    int mySum = sum(4, 5);
    cout << mySum << endl;
    return 0;
}
```

We define `sum` here
Before we call it down here
Function Order

• The order in which functions are defined matters in C++
• You cannot call a function before it’s been *defined* or *declared*

```cpp
int main() {
    int mySum = sum(4, 5);
    cout << mySum << endl;
    return 0;
}

int sum(int val1, int val2) {
    int result = val1 + val2;
    return result;
}
```

*We call `sum` here*

*But we don’t define it until down here… ERROR*
Function Order

• The order in which functions are defined matters in C++
• You cannot call a function before it’s been defined or declared

```cpp
int sum(int val1, int val2);

int main() {
    int mySum = sum(4, 5);
    cout << mySum << endl;
    return 0;
}

int sum(int val1, int val2) {
    int result = val1 + val2;
    return result;
}
```

Function declaration for `sum` ✅

All good, as long as the declaration happens before we call `sum`

Function definition for `sum`, can be written later
What gets printed? 🧐

```cpp
int doubleValue(int x) {
    x *= 2;
    return x;
}
int main() {
    int myValue = 5;
    int result = doubleValue(myValue);
    cout << "myValue: " << myValue << " result: " << result << endl;
    return 0;
}
```

myValue: ?? result: ??
int doubleValue(int x) {
    x *= 2;
    return x;
}

int main() {
    int myValue = 5;
    int result = doubleValue(myValue);
    cout << "myValue: " << myValue << " result: " << result << endl;
    return 0;
}
What gets printed?

```cpp
int doubleValue(int x) {
    x *= 2;
    return x;  // Callee function
}

int main() {
    int myValue = 5;
    int result = doubleValue(myValue);
    cout << "myValue: " << myValue << " " << endl;
    cout << "result: " << result << endl;
    return 0;  // Caller function
}
```

myValue: 5 result: 10
Passing by Value

• By default, we pass parameters to functions *by value*
• This means the callee function gets a copy of our variable
• Changes made to that parameter variable in the callee function won’t affect our variable in the caller function
Passing by Value

• By default, we pass parameters to functions *by value*
• This means the callee function gets a copy of our variable
• Changes made to that parameter variable in the callee function won’t affect our variable in the caller function

*We’ll learn another way to pass parameters later on!*
Control Flow
Way to Control the Flow

• Conditionals (if/else)
• Loops (for/while)
Way to Control the Flow

- Conditionals (if/else)
- Loops (for/while)
- These are used with a boolean expression:

<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>
Conditionals

```java
if (condition) {
    // code to execute if condition is true
}
```
Conditionals

```java
if (condition) {
    // code to execute if condition is true
}
```

*Note this syntax!*  
*We put the condition in parentheses and the conditional body in curly braces.*
Conditionals

if (condition) {
    // code to execute if condition is true
} else {
    // code to execute if the condition is false
}
Conditionals

// assuming age variable is already defined
if (age < 12) {
    cout << "Eligible for kids meal.";
} else {
    cout << "Must use regular menu.";
}
// assuming age variable is already defined
if (age < 12) {
    cout << "Eligible for kids meal.";
} else if (age > 65) {
    cout << "Eligible for senior discount.";
} else {
    cout << "Must use regular menu.";
}
While Loops

• “While this condition is true, do this”
• Use when you don’t know how many times you want to repeat

while (condition) {
    // code to repeat while condition is true
}
For Loops

• Use when you know how many times you want to repeat
• Typical for loop uses int counter \( i \) that starts at 0:

```cpp
for (int i = 0; i < 10; i++) {
    cout << i << endl;
}
```
For Loops

- Use when you know how many times you want to repeat
- Typical for loop uses int counter $i$ that starts at 0
- More generally, for loops take on this structure:

```java
for (initialization; condition; update) {
    // code to be repeated
}
```
For Loops

• Use when you know how many times you want to repeat
• Typical for loop uses int counter i that starts at 0
• More generally, for loops take on this structure:

  initialization; condition; update

  for (int i = 10; i <= 100; i += 10) {
    cout << i << endl;
  }
For Loops

- Use when you know how many times you want to repeat
- Typical for loop uses int counter i that starts at 0
- More generally, for loops take on this structure:

  initialization; condition; update

  for (int i = 10; i <= 100; i += 10) {
    cout << i << endl;
  }

Talk to your neighbor: what gets printed?
For Loops

• Use when you know how many times you want to repeat.
• Typical for loop uses int counter i that starts at 0.
• More generally, for loops take on this structure:

  for (initialization; condition; update) {
    // body of loop
  }

  for (int i = 10; i <= 100; i += 10) {
    cout << i << endl;
  }
Let’s write a program!

Try implementing with a while loop, then a for loop!