CS 106X, Lecture 2
C++ Functions and Strings

reading:

*Programming Abstractions in C++,* Chapters 2-3
Plan for Today

• C++ Functions
  – Syntax
  – Prototypes
  – Pass by reference
• Announcements
• Strings
  – Common functions and patterns
  – C strings vs. C++ strings
Plan for Today

• **C++ Functions**
  – Syntax
  – Prototypes
  – Pass by reference
• **Announcements**
• **Strings**
  – Common functions and patterns
  – C strings vs. C++ strings
Defining a function (2.3)

• A C++ **function** is like a Java **method**.

```cpp
return type
parameters (arguments)

type functionName(type name, type name, ..., type name) {
    statement;
    statement;
    ...
    statement;
    return expression;  // if return type is not void
}
```

• Calling a function:

```cpp
parameters (arguments)

functionName(value, value, ..., value);
```
// Prints out the lyrics for the popular “bottle song”
void bottlesOfPop(int count) {
    cout << count << " bottles of pop on the wall." << endl;
    cout << count << " bottles of pop." << endl;
    cout << "Take one down, pass it around, " << (count-1) << 
        " bottles of pop on the wall." << endl << endl;
}

int main() {
    for (int i = 99; i > 0; i--) {
        bottlesOfPop(i);
    }
    return 0;
}
99 bottles of pop on the wall.
99 bottles of pop.
Take one down, pass it around, 98 bottles of pop on the wall.
98 bottles of pop on the wall.
98 bottles of pop.
Take one down, pass it around, 97 bottles of pop on the wall.
97 bottles of pop on the wall.
97 bottles of pop.
Take one down, pass it around, 96 bottles of pop on the wall.
...
3 bottles of pop on the wall.
3 bottles of pop.
Take one down, pass it around, 2 bottles of pop on the wall.
2 bottles of pop on the wall.
2 bottles of pop.
Take one down, pass it around, 1 bottles of pop on the wall.
1 bottles of pop on the wall.
1 bottles of pop.
Take one down, pass it around, 0 bottles of pop on the wall.
// Returns the area of a circle with the given radius.
double circleArea(int r) {
    return 3.14 * r * r;
}

int main() {
    double a1 = circleArea(1);    // call the function
    double a2 = circleArea(3);    // call it again
    cout << "The area is " << a1 << "!!" << endl;
    return 0;
}
// Returns the area of a circle with the given radius.
double circleArea(int r) {
    return 3.14 * r * r;
}

int main() {
    double a1 = circleArea(1);   // call the function
    double a2 = circleArea(3);   // call it again
    double a3 = circleArea(3.1); // truncates!
    cout << "The area is " << a1 << "!!" << endl;
    return 0;
}
// Returns the area of a circle with the given radius.
double circleArea(int r) {
    return 3.14 * r * r;
}

// Functions can share names as long as the arguments differ
double circleArea(double r) {
    return 3.14 * r * r;
}

int main() {
    double a1 = circleArea(1);   // call the function
    double a2 = circleArea(3);   // call it again
    double a3 = circleArea(3.1); // ok!
    cout << "The area is " << a1 << "!" << endl;
    return 0;
}
Default parameters

• You can make a parameter optional by supplying a default value:
  – All parameters with default values must appear last in the list.

```cpp
// Prints a line of characters of the given width.
void printLine(int width = 10, char letter = '*') {
    for (int i = 0; i < width; i++) {
        cout << letter;
    }
    cout << endl;
}
...
printLine(7, '?');    // ????????
printLine(5);         // ****
printLine();          // **********
```
Plan for Today

• C++ Functions
  – Syntax
  – Prototypes
  – Pass by reference
• Announcements
• Strings
  – Common functions and patterns
  – C strings vs. C++ strings
• **Compiler error**: unable to find the `circleArea` function (!)
  – C++ reads the file from top to bottom (unlike some other languages)

```cpp
int main() {
    double a = circleArea(2.5); // call the function
    return 0;
}

double circleArea(double r) {
    return 3.14159265359 * r * r;
}
```
**Function prototypes (1.4)**

*type name(type name, type name, ..., type name);*

- Declare the function (without writing its body) at top of program.

```c
double circleArea(double r); // function prototype

int main() {
    double a = circleArea(2.5); // call the function
    return 0;
}

double circleArea(double r) {
    ...
}
```

*With prototype, only declare default values in prototype.*
Math functions (2.1)

- `#include <cmath>`

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description (returns)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>abs(value)</code></td>
<td>absolute value</td>
</tr>
<tr>
<td><code>ceil(value)</code></td>
<td>rounds up</td>
</tr>
<tr>
<td><code>floor(value)</code></td>
<td>rounds down</td>
</tr>
<tr>
<td><code>log10(value)</code></td>
<td>logarithm, base 10</td>
</tr>
<tr>
<td><code>max(value1, value2)</code></td>
<td>larger of two values</td>
</tr>
<tr>
<td><code>min(value1, value2)</code></td>
<td>smaller of two values</td>
</tr>
<tr>
<td><code>pow(base, exp)</code></td>
<td><code>base</code> to the <code>exp</code> power</td>
</tr>
<tr>
<td><code>round(value)</code></td>
<td>nearest whole number</td>
</tr>
<tr>
<td><code>sqrt(value)</code></td>
<td>square root</td>
</tr>
<tr>
<td><code>sin(value)</code></td>
<td>sine/cosine/tangent of an angle in radians</td>
</tr>
<tr>
<td><code>cos(value)</code></td>
<td></td>
</tr>
<tr>
<td><code>tan(value)</code></td>
<td></td>
</tr>
</tbody>
</table>

- unlike in Java, you don't write `Math.` in front of the function name
- see Stanford "gmath.h" library for additional math functionality
Plan for Today

• C++ Functions
  – Syntax
  – Prototypes
  – Pass by reference

• Announcements

• Strings
  – Common functions and patterns
  – C strings vs. C++ strings
void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}

int main() {
    int x = 17;
    int y = 35;
    swap(x, y);
    cout << x << "," << y << endl;   // 17,35
    return 0;
}
void swap(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}

int main() {
    int x = 17;
    int y = 35;
    swap(x, y);
    cout << x << "," << y << endl;  // 35,17
    return 0;
}
int main() {
    int x = 17;
    int y = 35;
    swap(x, y);
    cout << x << "," << y << endl;  // 35,17
    return 0;
}

– ‘&’ just in function definition, no change when calling function (hard to read)
– Can’t pass in literals (e.g. swap(1, 3) doesn’t work)
– Fast for large data types (e.g. Vector) – no copies made
– Allows for multiple changes to persist from a function
Quadratic exercise

• Write a function **quadratic** to find roots of quadratic equations.

\[ a x^2 + b x + c = 0, \text{ for some numbers } a, b, \text{ and } c. \]

– Find roots using the **quadratic formula**.

– Example: \( x^2 - 3 x - 4 = 0 \)

  roots: \( x = 4, \ x = -1 \)

\[
\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
\]

– What parameters should our function accept? What should it return?

  • Which parameters should be passed by value, and which by reference?
#include <math.h>
...

/*
* Solves a quadratic equation $ax^2 + bx + c = 0$,
* storing the results in output parameters root1 and root2.
* Assumes that the given equation has two real roots.
*/

void quadratic(double a, double b, double c,
                double& root1, double& root2) {
    double d = sqrt(b * b - 4 * a * c);
    root1 = (-b + d) / (2 * a);
    root2 = (-b - d) / (2 * a);
}

$$-b \pm \sqrt{b^2 - 4ac} \over 2a$$
• When solving a large problem, break down the task into functions.

• Properties of a good function:
  – Fully performs a single coherent task.
  – Does not do too large a share of the work.
  – Is not unnecessarily connected to other functions.
  – Stores data at the narrowest scope possible.

• The main function should be a concise summary of the overall program.
  – Most calls to other functions should be made by main.
int main() {
    double a, b, c, root1, root2;
    getCoefficients(a, b, c);
    solveQuadratic(a, b, c, root1, root2);
    printRoots(root1, root2);
    return 0;
}
Decomposition Example

int main() {
    double a, b, c, x, y;
    quadratic(a, b, c, x, y);
    return 0;
}

- main() is a poor program summary
- This function completes all program tasks
- This function handles all the data
- Functions and variables are poorly named
Plan for Today

• C++ Functions
  – Syntax
  – Prototypes
  – Pass by reference

• Announcements

• Strings
  – Common functions and patterns
  – C strings vs. C++ strings
Announcements

• Assignment 0 due **Friday 9/28 11AM**
  – If you don’t have your own computer, use library or cluster!
• Qt Creator Troubleshooting Session **tonight (Wed) 7-9PM @ LaIR**
• Required lecture feedback
• Discussion Section signups open **9/27 5PM – 9/30 5PM**
• CodeStepByStep 🍀
CURIS Poster Session

Friday 9/28 3-5PM on Packard Lawn

See awesome summer research projects and get a taste of what CS research is like!
WiCS Frosh Interns

WiCS Frosh Intern Program

{ Curious about CS? Looking for a community on campus? Excited about the WiCS mission? }

Apply for the WiCS Frosh Intern Program at bit.ly/wics-frosh-intern-1819

{ Frosh interns rotate through different WiCS teams, work on meaningful projects, and join a community of lifelong friends and mentors }

Applications are due Friday, Oct. 5 at 11:59 PM

Stanford Women in Computer Science {W}
# CS 581: Media Innovation

**CS 581**  
**MEDIA INNOVATION**  
**FALL: TUESDAYS NOON-1:30PM, 1 UNIT, CR/NC**  
**BROWN INSTITUTE FOR MEDIA INNOVATION**

<table>
<thead>
<tr>
<th>Course Title</th>
<th>Speaker(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>How to Build a Culture of Innovation in Media</em></td>
<td>Matter</td>
</tr>
<tr>
<td><em>Tech Trends for Media</em></td>
<td>Slack</td>
</tr>
<tr>
<td><em>Innovating in the Public Square</em></td>
<td>Princeton's J. Nathan Matias</td>
</tr>
<tr>
<td><em>Innovating with VR</em></td>
<td>Float</td>
</tr>
<tr>
<td><em>Rethinking Tech, Ethics + Democracy</em></td>
<td>MIT Technology Review</td>
</tr>
<tr>
<td><em>Twitch 4 News?</em></td>
<td>Twitch</td>
</tr>
<tr>
<td><em>On Algorithmic Bias</em></td>
<td>The Markup</td>
</tr>
<tr>
<td><em>Innovating with Drone Photography</em></td>
<td>Photo Technology Editor Josh Haner</td>
</tr>
<tr>
<td><em>Rethinking Audio</em></td>
<td>Google Home</td>
</tr>
</tbody>
</table>

**Come Hear + Learn from Industry Experts!**
Stanford Local Programming Contest

Stanford Local Programming Contest | 2018

October 6th, Sat, 9:00 am
Gates B08/B12/B30

Register at:
http://cs.stanford.edu/group/acm/SLPC

sponsored by Google
Plan for Today

• C++ Functions
  – Syntax
  – Prototypes
  – Pass by reference
• Announcements
• Strings
  – Common functions and patterns
  – C strings vs. C++ strings
#include <string>
...
string s = "hello";

• A string is a (possibly empty) sequence of characters.
• Strings are mutable (can be changed) in C++.
• There are two types of strings in C++.

:-/
Characters

- Characters are values of type `char`, with 0-based indexes:
  
  ```cpp
  string s = "Hi 106X!";
  
  char c1 = s[3];  // '1'
  char c2 = s.at(1);  // 'i'
  ```

- Characters have **ASCII** encodings (integer mappings):
  ```cpp
  cout << (int) s[0] << endl;  // 72
  ```
• #include <cctype>

– Useful functions to process char values (not entire strings):

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isalpha(c)</td>
<td>isalnum(c)</td>
</tr>
<tr>
<td>isdigit(c)</td>
<td>isspace(c)</td>
</tr>
<tr>
<td>isupper(c)</td>
<td>ispunct(c)</td>
</tr>
<tr>
<td>islower(c)</td>
<td></td>
</tr>
<tr>
<td>tolower(c)</td>
<td>toupper(c)</td>
</tr>
</tbody>
</table>

returns true if the given character is an alphabetic character from a-z or A-Z, a digit from 0-9, an alphanumeric character (a-z, A-Z, or 0-9), an uppercase letter (A-Z), a space character (space, \t, \n, etc.), or a punctuation character (., ; !), respectively

// index 012345678901234567890
string s = "Grace Hopper Bot v2.0";
if (isalpha(s[6]) && isnumber(s[18])
    && isspace(s[5]) && ispunct(s[19])) {
    cout << "Grace Hopper Smash!!" << endl;
}
Operators (3.2)

• **Concatenate** using + or += :

```java
string s1 = "Dai";
s1 += "sy"; // "Daisy"
```

• **Compare** using relational operators (ASCII ordering):

```java
string s2 = "Nick"; // == != < <= > >=
if (s1 < s2 && s2 != "Joe") { // true
    ...
}
```

• Strings are **mutable** and can be changed (!):

```java
s2.append(" Troccoli"); // "Nick Troccoli"
s2.erase(6, 7); // "Nick T"
s2[2] = '<'; // "Ni<k T"
```
### Member functions (3.2)

<table>
<thead>
<tr>
<th>Member function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s.append(str)</code></td>
<td>add text to the end of a string</td>
</tr>
<tr>
<td><code>s.compare(str)</code></td>
<td>return -1, 0, or 1 depending on relative ordering</td>
</tr>
<tr>
<td><code>s.erase(index, length)</code></td>
<td>delete text from a string starting at given index</td>
</tr>
<tr>
<td><code>s.find(str)</code></td>
<td>first or last index where the start of <code>str</code> appears in this string (returns string::npos if not found)</td>
</tr>
<tr>
<td><code>s.rfind(str)</code></td>
<td></td>
</tr>
<tr>
<td><code>s.insert(index, str)</code></td>
<td>add text into a string at a given index</td>
</tr>
<tr>
<td><code>s.length()</code> or <code>s.size()</code></td>
<td>number of characters in this string</td>
</tr>
<tr>
<td><code>s.replace(index, len, str)</code></td>
<td>replaces <code>len</code> chars at given index with new text</td>
</tr>
<tr>
<td><code>s.substr(start, length)</code> or</td>
<td>the next <code>length</code> characters beginning at <code>start</code></td>
</tr>
<tr>
<td><code>s.substr(start)</code></td>
<td>(inclusive); if <code>length</code> omitted, grabs till end of string</td>
</tr>
</tbody>
</table>

```cpp
string name = "Nick Troccoli";
if (name.find("Troccoli") != string::npos) {
    name.erase(6, 7);   // Nick T
}```
#include "strlib.h"

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>endsWith(str, suffix)</code></td>
<td>true if string begins or ends with the given text</td>
</tr>
<tr>
<td><code>startsWith(str, prefix)</code></td>
<td>true if string begins or ends with the given prefix</td>
</tr>
<tr>
<td><code>integerToString(int)</code></td>
<td>convert between numbers and strings</td>
</tr>
<tr>
<td><code>realToString(double)</code></td>
<td>convert between numbers and strings</td>
</tr>
<tr>
<td><code>stringToInteger(str)</code></td>
<td>convert between numbers and strings</td>
</tr>
<tr>
<td><code>stringToReal(str)</code></td>
<td>convert between numbers and strings</td>
</tr>
<tr>
<td><code>equalsIgnoreCase(s1, s2)</code></td>
<td>true if s1 and s2 have same chars, ignoring casing</td>
</tr>
<tr>
<td><code>toLowerCase(str)</code></td>
<td>returns an uppercase version of a string</td>
</tr>
<tr>
<td><code>toUpperCase(str)</code></td>
<td>returns an uppercase version of a string</td>
</tr>
<tr>
<td><code>trim(str)</code></td>
<td>returns string with surrounding whitespace removed</td>
</tr>
</tbody>
</table>

```c
if (startsWith(name, "Professor")) {
    name += " " + integerToString(workYears) + " years teaching";
}
```
String exercise

• Write a function `nameDiamond` that accepts a string parameter and prints its letters in a "diamond" format as shown below.
  – For example, `nameDiamond("DAISY")` should print:

```
D
DA
DAI
DAIS
DAISY
AISY
ISY
SY
Y
```
```cpp
void nameDiamond(string& name) {
    // print top half of diamond
    for (int i = 1; i <= name.length(); i++) {
        cout << name.substr(0, i) << endl;
    }

    // print bottom half of diamond
    for (int i = 1; i < name.length(); i++) {
        for (int j = 0; j < i; j++) { // indent
            cout << " "; // with spaces
        }
        cout << name.substr(i) << endl;
    }
}
```
String user input (3.1)

- `cin` reads string input, but only a word at a time:
  ```cpp
  cout << "Type your name: ";
  string name; // Type your name: John Doe
  cin >> name;  // Hello, John
  cout << "Hello, " << name << endl;
  ```

- Stanford library `getLine` function reads an entire line:
  ```cpp
  string name = getline("Type your name: ");
  cout << "Hello, " << name << endl;  // Hello, John Doe
  ```

- C++ standard lib `getline` function is similar:
  ```cpp
  string name;
  cout << "Type your name: ";
  getline(cin, name);
  cout << "Hello, " << name << endl;
  ```
Plan for Today

• C++ Functions
  – Syntax
  – Prototypes
  – Pass by reference

• Announcements

• Strings
  – Common functions and patterns
  – C strings vs. C++ strings
C vs. C++ strings (3.5)

• C++ has two kinds of strings:
  – **C strings** (char arrays) and **C++ strings** (string objects)

• A string literal such as "hi there" is a C string.
  – C strings don't include any methods/behavior shown previously.
    • No member functions like length, find, or operators.

• Converting between the two types:
  – `string("text")` C string to C++ string
  – `string.c_str()` C++ string to C string
C string bugs

- `string s = "hi" + "there";  // C-string + C-string`
- `string s = "hi" + '?';       // C-string + char`
- `string s = "hi" + 41;       // C-string + int`
  - C strings can't be concatenated with +.
  - C-string + char/int produces garbage, not "hi?" or "hi41".
  - This bug usually appears in print statements, and you'll see partial strings.

- `string s = "hi";
  s += 41;                    // "hi")`
  - Adds character with ASCII value 41, ')', doesn't produce "hi41".

- `int n = (int) "42";        // n = 0x7ffdcb08`
  - Bug; sets n to the memory address of the C string "42" (ack!).
C string bugs fixed

• string s = string("hi") + "there";
• string s = "hi"; // convert to C++ string
  s += "there";
  – These both compile and work properly.

• string s = "hi"; // C++ string + char
  s += '?'; // "hi?"
  – Works, because of auto-conversion.

• s += integerToString(41); // "hi?41"
• int n = stringToInteger("42"); // 42
  – Explicit string <-> int conversion using Stanford library.
Recap

• C++ Functions
  – Syntax
  – Prototypes
  – Pass by reference

• Announcements

• Strings
  – Common functions and patterns
  – C strings vs. C++ strings

Next time: C++ file reading and Grids
Overflow (extra) slides
• What if you want to avoid copying a large variable but don't want to change it?
• Use the `const` keyword to indicate that the parameter won't be changed
  – Usually used with strings and collections
  – Passing in a non-variable (e.g. `printString("hello")`) **does** work

```cpp
void printString(const string& str) {
    cout << "I will print this string" << endl;
    cout << str << endl;
}

int main() {
    printString("This could be a really really long string");
}
```
What is the output of this code?

```c++
void mystery(int& b, int c, int& a) {
    a++;
    b--;
    c += a; // A. 5 2 8 // B. 5 3 7 // C. 6 1 8
}

int main() {
    int a = 5; // D. 6 1 13 // E. other
    int b = 2;
    int c = 8;
    mystery(c, a, b);
    cout << a << " " << b << " " << c << endl;
    return 0;
}
```
What is the output of the following program?

```c
int mystery(int b, int c) {
    return c + 2 * b;
}

int main() {
    int a = 4;
    int b = 2;
    int c = 5;
    a = mystery(c, b);
    c = mystery(b, a);
    cout << a << " " << b << " " << c << endl;
    return 0;
}
```

// A. 12 2 16
// B. 9 2 10
// C. 12 2 8
// D. 9 2 12
// E. N/A
Exercise: BMI

• Write code to calculate 2 people's body mass index (BMI):

\[BMI = \frac{weight}{height^2} \times 703\]

• Match the following example output:

This program reads data for two people and computes their Body Mass Index (BMI).

Enter Person 1's information:
height (in inches)? 70.0
weight (in pounds)? 194.25
BMI = 27.8689, class 3

Enter Person 2's information:
height (in inches)? 62.5
weight (in pounds)? 130.5
BMI = 23.4858, class 2

BMI difference = 4.3831

<table>
<thead>
<tr>
<th>BMI</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>below 18.5</td>
<td>class 1</td>
</tr>
<tr>
<td>18.5 - 24.9</td>
<td>class 2</td>
</tr>
<tr>
<td>25.0 - 29.9</td>
<td>class 3</td>
</tr>
<tr>
<td>30.0 and up</td>
<td>class 4</td>
</tr>
</tbody>
</table>
/* Prints a welcome message explaining the program. */
void introduction() {
    cout << "This program reads data for two people" << endl;
    cout << "and computes their body mass index (BMI)." << endl << endl;
}

/* Computes/returns a person's BMI based on their height and weight. */
double computeBMI(double height, double weight) {
    return weight * 703 / height / height;
}

/* Outputs information about a person's BMI and weight status. */
int bmiClass(double bmi) {
    if (bmi < 18.5) {
        return 1;
    } else if (bmi < 25) {
        return 2;
    } else if (bmi < 30) {
        return 3;
    } else {
        return 4;
    }
}
/* Reads information for one person, computes their BMI, and returns it. */
double person(int number) {
    cout << "Enter person " << number << "'s information:" << endl;
    double height = getReal("height (in inches)? ");
    double weight = getReal("weight (in pounds)? ");
    double bmi = computeBMI(height, weight);
    cout << "BMI = " << bmi << ", class " << bmiClass(bmi) << endl << endl;
    return bmi;
}

/* Main function to run the overall program. */
int main() {
    introduction();
    double bmi1 = person(1);
    double bmi2 = person(2);
    cout << "BMI difference = " << abs(bmi1 - bmi2) << endl;
    return 0;
}
void mystery(string a, string& b) {
    a.erase(0, 1); // erase 1 from index 0
    b += a[0];    // insert at index 3
    b.insert(3, "FOO"); // insert at index 3
}

int main() {
    string a = "nick";  // A. nick troFO0ccoli
    string b = "troccoli"; // B. nick troccoli
    mystery(a, b);      // C. nick troccoliFOO
    cout << a << " " << b << endl; // D. nickFOO troccoli
    return 0;
} // E. nick troFO0ccoli