Namespaces and using

- **namespace**: An area for scoping identifiers (functions, variables).
  - Helps avoid collisions between items with the same name.
  - C++ console I/O objects (cout, cin, etc.) are in name space `std` .

- `using namespace` ***`name`*****`;`**
  - Brings symbols from a library's "name space" into the global scope of your program so you can refer to them.

- ***`namespace`*****`::`*****`identifier`**
  - without a `using` declaration, you can access symbols from a namespace by preceding them with their namespace name and `::`.

```
std::cout << "Hello, world!" << std::endl;
```

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

reading:

Programming Abstractions in C++, Chapters 2-3

This document is copyright (C) Stanford Computer Science and Nick Troccoli, licensed under Creative Commons Attribution 2.5 License. All rights reserved.
Based on slides created by Keith Schwarz, Julie Zelenski, Marty Stepp, Ashley Taylor, Jerry Cain, Eric Roberts, Mehran Sahami, Stuart Reges, Cynthia Lee, and others.

Plan for Today

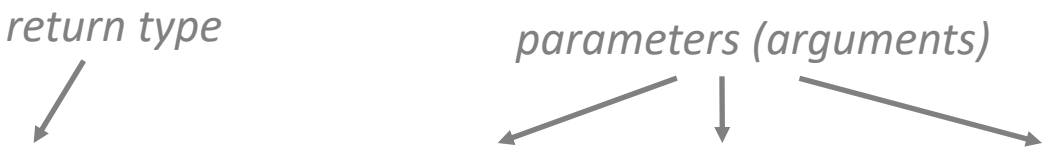
- C++ Functions
  - Syntax
  - Prototypes
  - Pass by reference
- Announcements
- Strings
  - Common functions and patterns
  - C strings vs. C++ strings

Plan for Today

- C++ Functions
  - Syntax
  - Prototypes
  - Pass by reference
- Announcements
- Strings
  - Common functions and patterns
  - C strings vs. C++ strings

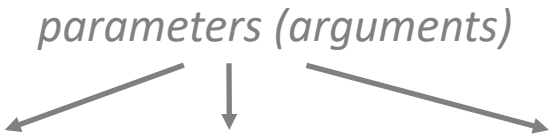
Defining a function (2.3)

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




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

- Calling a function:




```
functionName(value, value, ..., value);
```

Defining a function

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

// 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;
}
```

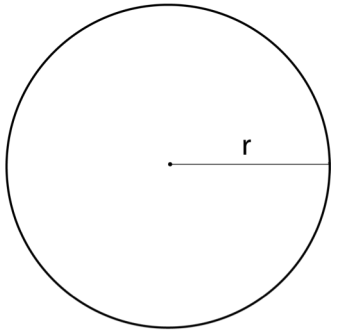
Lots of Pop

```
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.
```

Defining a function

```
// Returns the area of a circle with the given radius.
double circleArea(int r) {
    return 3.14 * r * r;
}

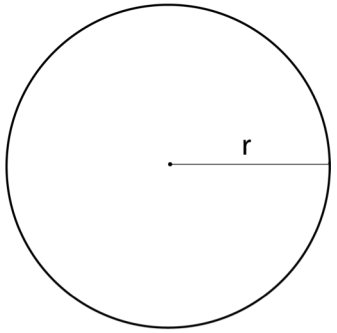
int main() {
    double a1 = circleArea(1);   // call the function
    double a2 = circleArea(3);   // call it again
    cout << "The area is " << a1 << "!!" << endl;
    return 0;
}
```



Defining a function

```
// 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;
}
```

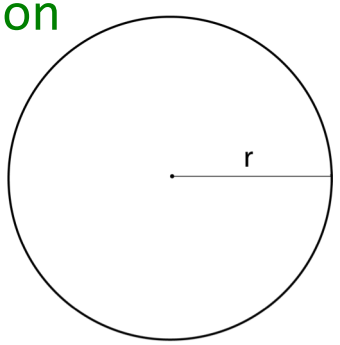


Overloading

```
// 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.

```
// 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

Declaration order

- **Compiler error**: unable to find the `circleArea` function (!)
  - C++ reads the file from top to bottom (unlike some other languages)

```
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.

```
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>`

Function name	Description (returns)
`abs(`***value***`)`	absolute value
`ceil(`***value***`)`	rounds up
`floor(`***value***`)`	rounds down
`log10(`***value***`)`	logarithm, base 10
`max(`***value1, value2***`)`	larger of two values
`min(`***value1, value2***`)`	smaller of two values
`pow(`***base, exp***`)`	*base* to the *exp* power
`round(`***value***`)`	nearest whole number
`sqrt(`***value***`)`	square root

`sin(`***value***`)`
`cos(`***value***`)`
`tan(`***value***`)`
	sine/cosine/tangent of an angle in radians

  - 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

Pass by Value

```
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;
}
```

By default, C++ parameters are copies.

Pass by Reference

```
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;
}
```

Use “&” to pass the same version.

Pass by Reference

```
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

Output parameters

xkcdDatingRange

- What is the minimum and maximum non-creepy age to date?

```
void datingRange(int age, int& min, int& max) {
    min = age / 2 + 7;
    max = (age - 7) * 2;
}

int main() {
    int young;
    int old;
    datingRange(48, young, old);
    count << "A 48-year-old could date someone from "
          << young << " to " << old " years old." << endl;
}

// A 48-year-old could date someone from
// 31 to 82 years old.
```

young and old 无需初始化即可赋值

http://xkcd.com/314/

design gets set up here
when min refers to young

Reference pros/cons

- **benefits** of reference parameters:
  - a useful way to be able to 'return' more than one value
  - often used with objects, to avoid making bulky copies when passing

大数据结构，stream常用pass by reference

- **downsides** of reference parameters:
  - hard to tell from call whether it is ref; can't tell if it will be changed
    - `foo(a, b, c);  // will foo change a, b, or c?  :-/`
  - slightly slower than value parameters
  - can't pass a literal value to a ref parameter; must "refer" to a variable
    - `grow(39);  // error`

不能放literal进去，必须指向一个变量

this code if i pass
boo abc

Quadratic exercise

- Write a function **`quadratic`** to find roots of quadratic equations.

  $a\,x^2 + b\,x + c = 0$, for some numbers a, b, 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?

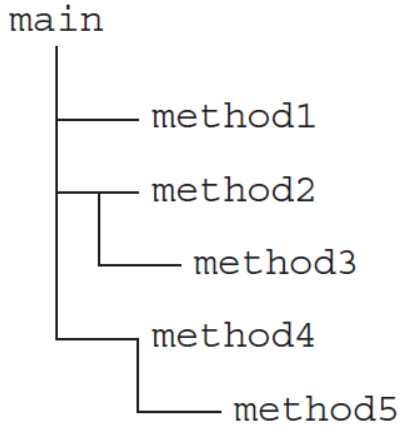
Quadratic solution

```
#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);
}
```

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

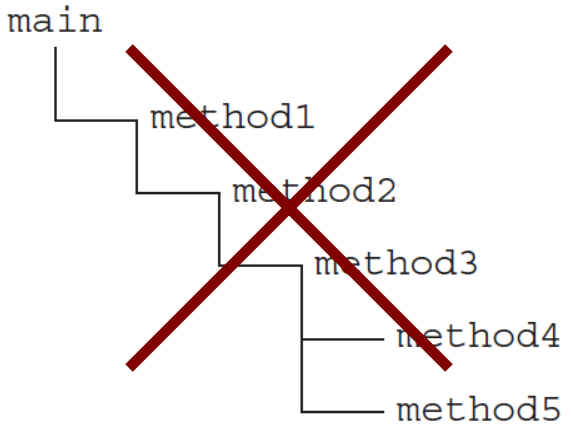
Decomposition

- 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`.



Decomposition Example

```
int main() {
    double a, b, c, root1, root2;
    getCoefficients(a, b, c);
    solveQuadratic(a, b, c, root1, root2);
    printRoots(root1, root2);
    return 0;
}
```

- main() is a clear program summary
- Each function completes a discrete subtask
- Each function handles a subset of data
- Functions and variables are well named

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 

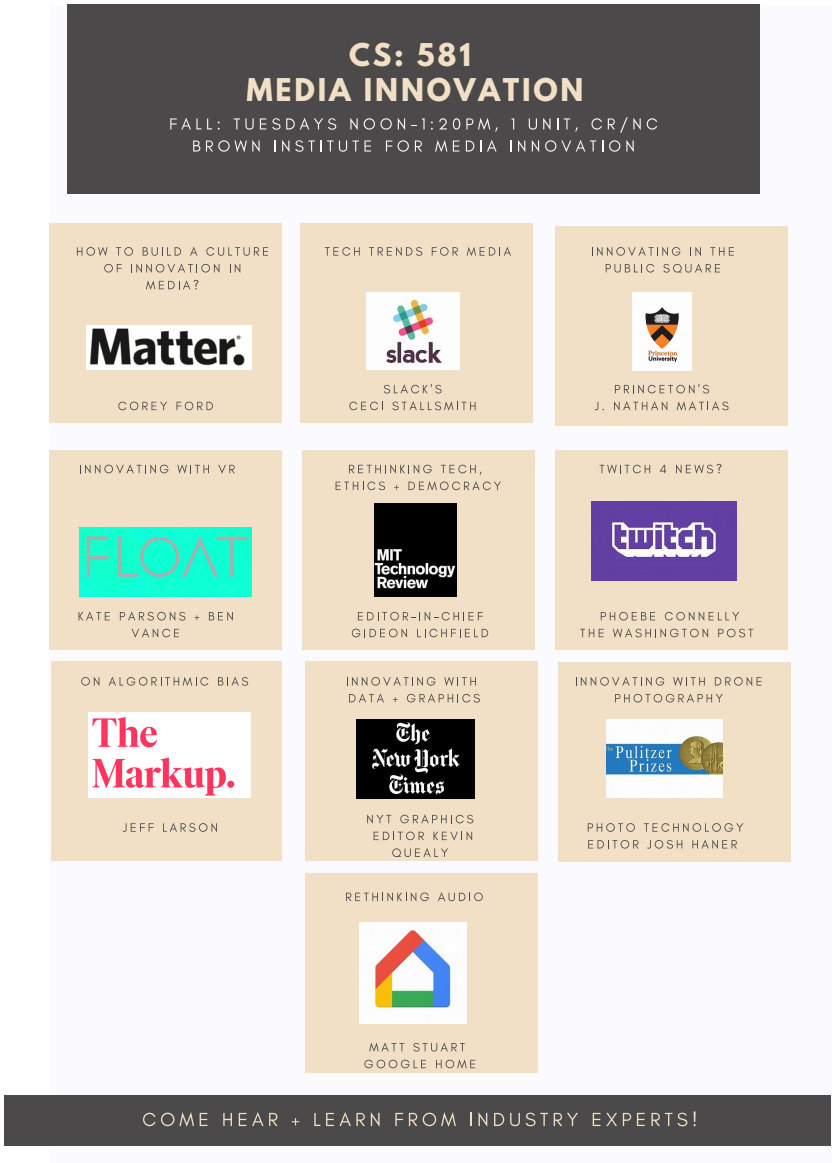
CURIS Poster Session

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

CS 581: Media Innovation



Stanford Local Programming Contest

Stanford Local Programming Contest | 2018

icpc.foundation icpc International Collegiate Programming Contest

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

Strings (3.1)

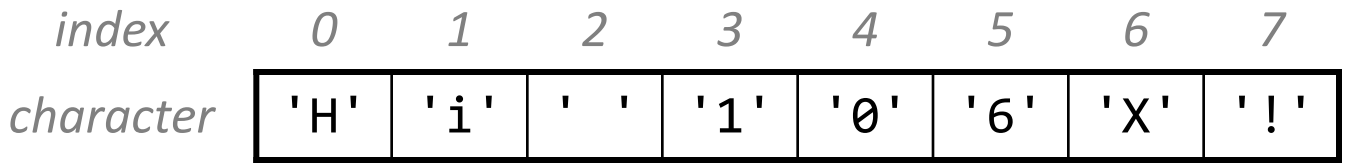
```
#include <string>
...
string s = "hello";
```

- 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:

```
string s = "Hi 106X!";
```



- Individual characters can be accessed using `[`***index***`]` or `at`:

```
char c1 = s[3];                 // '1'
char c2 = s.at(1);              // 'i'
```

- Characters have **ASCII** encodings (integer mappings):

```
cout << (int) s[0] << endl;    // 72
```

Char and cctype (3.3)

- `#include <cctype>`
  - Useful functions to process `char` values (not entire strings):

Function name	Description
`isalpha(c)` `isalnum(c)` `isdigit(c)` `isspace(c)` `isupper(c)` `ispunct(c)` `islower(c)`	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
`tolower(c)` `toupper(c)`	returns lower/uppercase equivalent of a character

```
//    index 01234567890123456789
string s = "Grace Hopper Bot v2.0";
if (isalpha(s[6]) && isnumer(s[18])
        && isspace(s[5]) && ispunct(s[19])) {
    cout << "Grace Hopper Smash!!" << endl;
}
```

Operators (3.2)

- **Concatenate** using + or += :

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

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

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

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

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

Member functions (3.2)

Member function name	Description
`s.append(str)`	add text to the end of a string
`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)` `s.rfind(str)`	first or last index where the start of ***str*** appears in this string (returns `string::npos` if not found)
`s.insert(index, str)`	add text 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 given index with new text
`s.substr(start, length)` or `s.substr(start)`	the next *length* characters beginning at *start* (inclusive); if *length* omitted, grabs till end of string

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

Stanford library (3.7)

- `#include "strlib.h"`

Function name	Description
`endsWith(`***str, suffix***`)` `startsWith(`***str, prefix***`)`	true if string begins or ends with the given text

`integerToString(`***int***`)`
`realToString(`***double***`)`
`stringToInteger(`***str***`)`
`stringToReal(`***str***`)`
	convert between numbers and strings

`equalsIgnoreCase(`***s1, s2***`)`	true if s1 and s2 have same chars, ignoring casing
`toLowerCase(`***str***`)` `toUpperCase(`***str***`)`	returns an upper/lowercase version of a string
`trim(`***str***`)`	returns string with surrounding whitespace removed

```
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
```

Exercise solution

```
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;
    }
}
```

```
D
DA
DAI
DAIS
DAISY
```

```
 AISY
  ISY
   SY
    Y
```

String user input (3.1)

- `cin` reads string input, but only a word at a time:

```
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:

```
string name = getLine("Type your name: ");
cout << "Hello, " << name << endl;  // Hello, John Doe
```

- C++ standard lib **`getline`** function is similar:

```
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

This document is copyright (C) Stanford Computer Science and Nick Troccoli, licensed under Creative Commons Attribution 2.5 License. All rights reserved.
Based on slides created by Keith Schwarz, Julie Zelenski, Marty Stepp, Ashley Taylor, Jerry Cain, Eric Roberts, Mehran Sahami, Stuart Reges, Cynthia Lee, and others.

Const parameters

- 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

```
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");
}
```

Ref param mystery

parameterMysteryBCA

- What is the output of this code?

```
void mystery(int& b, int c, int& a) {
    a++;
    b--;
    c += a;
}

int main() {
    int a = 5;
    int b = 2;
    int c = 8;
    mystery(c, a, b);
    cout << a << " " << b << " " << c << endl;
    return 0;
}
```

```
// A. 5 2 8
// B. 5 3 7
// C. 6 1 8
// D. 6 1 13
// E. other
```

Return mystery

returnMystery1

- What is the output of the following program?

```
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.          B.          C.          D.          E.
    //  12 2 16     9 2 10      12 2 8      9 2 12      N/A
```

Exercise: BMI

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

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

BMI	Category
below 18.5	class 1
18.5 - 24.9	class 2
25.0 - 29.9	class 3
30.0 and up	class 4

- 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
```

info
enter info
calculate,
classify
difference

BMI solution

```
/* 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;
    }
}
```

BMI solution, cont'd

```
/* 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;
}
```

What's the output?

stringMysteryAB

```
void mystery(string a, string& b) {
    a.erase(0, 1);          // erase 1 from index 0
    b += a[0];
    b.insert(3, "FOO");     // insert at index 3
}

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