

CS 106B, Lecture 2

Functions and Strings

reading:

Programming Abstractions in C++, Chapters 2-3

Plan for Today

- Functions
 - Syntax
 - Prototypes
 - Pass by value vs. reference; the const keyword
- Strings
 - Common functions and manipulations
 - C vs. C++ strings

Defining functions (2.3)

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

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:
parameters (arguments)
↙ ↓ ↘
functionName(*value*, *value*, ..., *value*);

Defining a function

```
#include "console.h"
using namespace std;
const string DRINK_TYPE = "Coke";

// Function Definition and Code
void bottles(int count) {
    cout << count << " bottles of " << DRINK_TYPE << " on the wall." << endl;
    cout << count << " bottles of " << DRINK_TYPE << "." << endl;
    cout << "Take one down, pass it around, " << (count-1) <<
        " bottles of " << DRINK_TYPE << " on the wall." << endl << endl;
}

int main() {
    for (int i = 99; i > 0; i--) {
        bottles(i);
    }
    return 0;
}
```

Declaration order

- **Compiler error:** unable to find the `bottles` function (!)
 - C++ reads the file from top to bottom (unlike Java or Python)

```
int main() {
    for (int i = 99; i > 0; i--) {
        bottles(i);
    }
    return 0;
}
```

```
void bottles(int count) {
    cout << count << " bottles of " << DRINK_TYPE << " on the wall." << endl;
    cout << count << " bottles of " << DRINK_TYPE << "." << endl;
    cout << "Take one down, pass it around, " << (count-1) <<
        " bottles of " << DRINK_TYPE << " on the wall." << endl << endl;
}
```

Function prototypes (1.4)

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

- Declare the function (without writing its body) at top of program.
- Include everything up to the first curly brace

```
void bottles(int count); // Function prototype
```

```
int main() {  
    for (int i = 99; i > 0; i--) {  
        bottles(i);  
    }  
    return 0;  
}
```

```
void bottles(int count) {  
    cout << count << " bottles of " << DRINK_TYPE << " on the wall." << endl;  
    cout << count << " bottles of " << DRINK_TYPE << "." << endl;  
    cout << "Take one down, pass it around, " << (count-1) <<  
        " bottles of " << DRINK_TYPE << " on the wall." << endl << endl;  
}
```

Pass by Value

- **value semantics:** In Java and C++, when variables (`int`, `double`) are passed as parameters, their values are copied.
 - Modifying a parameter will not affect the variable passed in.

```
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;  
}
```

Pass by Reference

- **reference semantics:** If you declare a parameter with an & after its type, it will link the caller and callee function variables to the same place in memory.

- Modifying a parameter *will* affect the variable passed in.
- The ampersand is only used in declaration, not in function call
- **Can't** pass in non-variables (e.g. `swap(1, 3)` won't work)
- Slightly slower for primitive (e.g. `int`) types
- Faster for larger types with many elements

```
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;  
}
```


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");  
}
```

Output parameters



xkcdDatingRange

- Can also pass by reference to return multiple items
- 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);  
    cout << "A 48-year-old could date someone from "  
        << young << " to " << old << " years old." << endl;  
}
```



<http://xkcd.com/314/>

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

Quadratic exercise



quadratic

- 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?

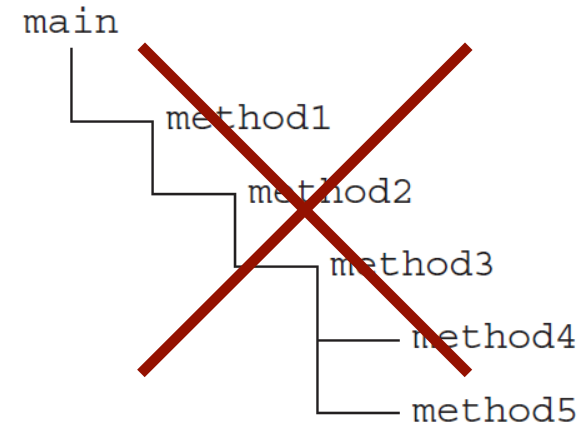
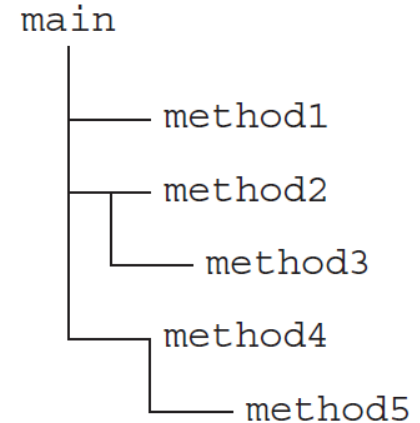
Quadratic solution

```
/*  
 * 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);  
}
```

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

Good Decomposition

- 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.
 - No "chaining" of functions
- The **main** function should be a concise summary of the overall program.
 - Basically an overview of the steps needed to solve the problem



Announcements

- Assignment 0 due Friday
 - Fill out the exam survey by **5PM on Friday**
 - Qt Creator Installation help session **today from 1:30-3:30PM** in Gates B02
- Undergraduate students should be enrolled for **5 units**
- Please make Piazza posts **public** as much as possible
- Sign up for section at cs198.stanford.edu
 - Section signups close **today at 5PM**
 - Make sure you sign up for the same sections as your partner (if you have one)

Strings

Click to add Text

Strings (3.1)

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

- A string is a (possibly empty) sequence of characters.
 - Minor differences from Java:
 - Different names/behavior for some member functions.
 - 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 106B!";
```

<i>index</i>	0	1	2	3	4	5	6	7
<i>character</i>	'H'	'i'	' '	'1'	'0'	'6'	'B'	'!'

- 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
```

Member functions (3.2)

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

```
string name = "Donald Knuth";  
if (name.find("Knu") != string::npos) {  
    name.erase(7, 5);           // "Donald"  
}
```

Operators (3.2)

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

```
string s1 = "Ash";  
s1 += "ley"; // "Ashley"
```

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

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

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

```
s1.append(" Taylor") // "Ashley Taylor"  
s1.erase(3, 3); // "Ash Taylor"  
s1[5] = '@'; // "Ash T@ylor"
```

Stanford library (3.7)

- `#include "strlib.h"`

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

```
if (startsWith(name, "Mr. ")) {  
    name += integerToString(age) + " years old";  
}
```

What's the output?

```
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
}

int main() { // 01234
    string a = "ashley";
    string b = "taylor";
    mystery(a, b);
    cout << a << " " << b << endl;
    return 0;
}
```

// A. shley taylor
// B. ashley taylor
// C. shley ataylorFOO
// D. ashley tayFOOlors
// E. shley tayFOolors

String exercise



nameDiamond

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

```
S
SH
SHR
SHRE
SHREY
SHREYA
 HREYA
  REYA
   EYA
    YA
     A
```

Exercise solution

```
void nameDiamond(string s) {
    int len = s.length();

    // print top half of diamond
    for (int i = 1; i <= len; i++) {
        cout << s.substr(0, i) << endl;
    }

    // print bottom half of diamond
    for (int i = 1; i < len; i++) {
        for (int j = 0; j < i; j++) { // indent
            cout << " ";           // with spaces
        }
        cout << s.substr(i, len - i) << endl;
    }
}
```

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 manifests 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 = 0x7ffdc08`
 - 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.
 - `string a = ...`
- `s += integerToString(41); // "hi?41"`
- `int n = stringToInteger("42"); // 42`
 - Explicit string <-> int conversion using Stanford library.

Overflow (extra) slides

Ref param mystery



- 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

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;
    }
}
```

...

```
printLine(7, '?');    // ???????
printLine(5);        // *****
printLine();         // *****************
```

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

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;
}
```

Char and ctype (3.3)

- #include <ctype>

– 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 012345678901234567890
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;
}
```