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
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

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

```cpp
return type

parameters (arguments)

typedef 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);
```
#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;
}

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)

```c++
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

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

```cpp
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;
}
• **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.

```java
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)
  - Faster for larger types with many elements

```c
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;
}
```
• 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 really long string");
}
```
Output parameters

• Can also pass by reference to return multiple items

• What is the minimum and maximum non-creepy age to date?
  
  ```cpp
  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;
  }
  ```

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

  [XKCD comic: Standard Creepiness Rule: Don't date under (age/2 + 7)](http://xkcd.com/314/)
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: \( x^2 - 3x - 4 = 0 \)
      
      roots: \( x = 4, x = -1 \)

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

  – 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);
}

\[-b \pm \sqrt{b^2 - 4ac} \over 2a\]
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
Plan for Today

• Functions
  – Syntax
  – Prototypes
  – Pass by value vs. reference; the const keyword

• Strings
  – Common functions and manipulations
  – C 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 indices:

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

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>character</td>
<td>'H'</td>
<td>'i'</td>
<td>' '</td>
<td>'1'</td>
<td>'0'</td>
<td>'6'</td>
<td>'B'</td>
<td>'!'</td>
</tr>
</tbody>
</table>

• 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

<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 &lt;0, 0, or &gt;0 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 <code>string::npos</code> 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 <code>s.substr(start)</code></td>
<td>the next <code>length</code> characters beginning at <code>start</code> (inclusive); if <code>length</code> omitted, grabs till end of string</td>
</tr>
</tbody>
</table>

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

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

  ```java
  string s1 = "Ty";
  s1 += "ler";        // "Tyler"
  ```

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

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

- **Strings are mutable** and can be changed:

  ```java
  s1.append(" Jay")        // "Tyler Jay"
  s1.erase(1, 3);          // "Tr Jay"
  s1[4] = '@';             // "Tr J@y"
  ```
# Stanford library

- `#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></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></td>
</tr>
<tr>
<td><code>stringToInteger(str)</code></td>
<td></td>
</tr>
<tr>
<td><code>stringToReal(str)</code></td>
<td></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 upper/lowercase version of a string</td>
</tr>
<tr>
<td><code>toUpperCase(str)</code></td>
<td></td>
</tr>
<tr>
<td><code>trim(str)</code></td>
<td>returns string with surrounding whitespace removed</td>
</tr>
</tbody>
</table>
What's the output?

```cpp
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";        // B. ashley taylor
    string b = "taylor";        // C. shley ataylorFOO
    mystery(a, b);              // D. ashley tayFOOlors
    cout << a << " " << b << endl;
    return 0;
}
```

// A. shley taylor
// B. ashley taylor
// C. shley ataylorFOO
// D. ashley tayFOOlors
// E. shley tayFoolors
• 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
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 = 0; i <= len; i++) {
        for (int j = 0; j < i; j++) {
            // indent
            cout << " ";
            // with spaces
        }
        cout << s.substr(i) << endl;
    }
}
C vs. C++ strings

• 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 = 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.
Look Ahead

• Assignment 0 due Thursday
  – Note: I had to make a few changes to the starter code. If you downloaded the ZIP file before 1:40PM Monday, please download it again
  – Qt Creator Installation help session tomorrow from 8-10pm in Gates B02

• Sign up for section at cs198.stanford.edu
  – Section signups close today at 5PM
Extra slides
• 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;
    int b = 2;
    int c = 8;
    mystery(c, a, b);
    cout << a << " " << b << " " << c << endl;  // D. 6 1 13
    return 0;
    // E. other
}
```
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
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;
    }
}
...
printLine(7, '?');  // ???????
printLine(5);      // ****
printLine();       // **********
```
Exercise: BMI

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

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

• 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
/* 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;
}
# Char and cctype

- **#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>tolower(c)</td>
</tr>
<tr>
<td></td>
<td>toupper(c)</td>
</tr>
</tbody>
</table>

```cpp
#include <iostream>
#include <cctype>

int main() {
    std::string s = "Grace Hopper Bot v2.0";
    if (isalpha(s[6]) && isalnum(s[18])
        && isspace(s[5]) && ispunct(s[19])) {
        std::cout << "Grace Hopper Smash!!\n" << std::endl;
    }
    return 0;
}
```