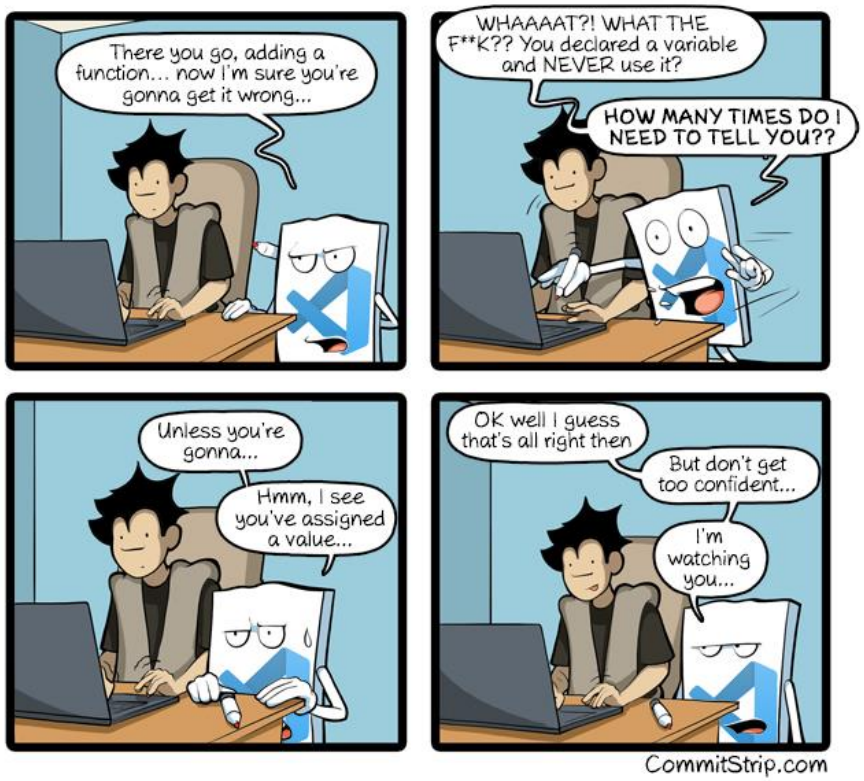
CS 132, Winter 2022

Lecture 3: `strings`; references



Thank you to Marty Stepp and Stuart Reges for parts of these slides

String exercise

- Write a function `equalsIgnoreCase` that accepts two strings as parameters and returns `true` if they are the same ignoring case, `false` otherwise.
  - For example, `equalsIgnoreCase("computer", "computep")` should return `false`.

char and cctype

- `#include <cctype>`
  - the same functions we used in C
  - notice, no `.h` in the include

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;
}
```

C vs. C++ strings

- C++ has a string type but we can still use `char*`/`char[]`:
  - **C strings** (`char` arrays) and **C++ strings** (`string` objects)

- A string literal such as `"hi there"` is a C string.
  - This means we can't use any methods/behavior shown previously on them.
    - Example: `"cat".length()` will not work

- Converting between the two types:
  - `string("`***`text`***`")` `// C string to C++ string`
  - ***`string`***`.c_str()` `// C++ string to C string`

Beware: C strings are still C strings

- `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"`.

- `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 += to_string(41);  // "hi?41"`
  `int n = stoi("42");  // 42`
  - Explicit string <-> int conversion.

Exercise solution

```
string toLowerCase(string s){
    if(s.length() == 0) {
        return string("");
    } else {
        char lower = tolower(s[0]);
        string result = toLowerCase(s.substr(1));
        return lower + result;
    }
}

bool equalsIgnoreCase(string s1, string s2) {
    return toLowerCase(s1) == toLowerCase(s2);
 }
```

String exercise

- Write a function `trim` that accepts a string as a parameter and removes all the whitespace at the beginning and the end of it. Leave any whitespace in the middle in the string.
  - For example, after the following code is run

```
string saying = "   computer s   ";
trim("   computer s   ");
```

`saying` should store `"computer s"`.

Why doesn't this work?

```
void removeFirst(string s) {
    s = s.substr(1);
}
```

- **C++ strings are passed by value** unless you specify otherwise
  - Solutions?
    - return a new copy
    - pass in a pointer
    - pass in a reference

An easier way

- C++ has an additional way to pass parameters: references
  - References
    - Basically, a pointer that C++ will auto dereference
  - Subtle differences:
    - A reference must be initialized when declared, this is optional with a pointer.
    - References can't be assigned to `NULL` but pointers can.
    - Pointers have to be dereferenced with a *, references don't.
    - Pointers can be changed to point at other values, references can't.

Reference semantics

- **reference semantics**: If you declare a parameter with an `&` after its type, it will make it a reference and so link the functions to the same place in memory.
  - Modifying a parameter *will* 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;    // 35,17
    return 0;
}
```

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
```

Output parameters

- 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.
```

http://xkcd.com/314/

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

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

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

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

- **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 for small types
  - can't pass a literal value to a ref parameter; must "refer" to a variable
    - `grow(39);   // error`

File Processing

`ifstream` members

Member function	Description
`f.clear();`	resets stream's error state, if any
`f.close();`	stops reading file
`f.eof()`	returns `true` if stream is *past* end-of-file (EOF)
`f.fail()`	returns `true` if the last read call failed (e.g. EOF)
`f.good()`	returns `true` if the file exists
`f.get()`	reads and returns one character
`f.open("filename");` `f.open(s.c_str());`	opens file represented by given C string (may need to write `.c_str()` if a C++ string is passed)
`f.unget(ch)`	un-reads one character
`f >> var`	reads data from input file into a variable (like `cin`); reads one whitespace-separated token at a time
`getline(f&, s&)`	reads line of input into a `string` by reference; returns a `true`/`false` indicator of success

Line-based I/O

- **Common pattern**: open a file; read each line; close it

```
// read and print every line of a file
#include <fstream>
...

ifstream input;
input.open("poem.txt");
string line;
while (getline(input, line)) {
    cout << line << endl;
}
input.close();
```

```
// incorrect (why?)
while (!input.fail()) {
    string line;
    getline(input, line);
    cout << line << endl;
}
```

Word-based I/O: >>

- Reads / converts next whitespace-separated token of input.
  - If unsuccessful, sets stream into *fail* state, and returns a "falsey" value.

```
ifstream input;
input.open("data.txt");
string word;
input >> word;    // "Percy"
input >> word;    // "is"
int age;
input >> age;     // 12
input >> word;    // "'years'"
input >> word;    // "old!"

if (input >> word) {    // false
    cout << "successful!" << endl;
}
```

data.txt:

```
Percy
 is   12
  'years'
  old!
```

Tokenizing a line

To use: `#include <sstream>`

- An **`istringstream`** lets you tokenize a string.

```
// read specific word tokens from a string
istringstream input("Jenny Smith 8675309");
string first, last;
int phone;
input >> first >> last;    // first="Jenny", last="Smith"
input >> phone;            // 8675309

// read all tokens from a string
istringstream input2("To be or not to be");
string word;
while (input2 >> word) {
    cout << word << endl;    // To \n be \n or \n not \n ...
}
```

Building a string

To use: `#include <sstream>`

- An **`ostringstream`** lets you write output into a string buffer.
  - Use the **`str`** method to extract the string that was built.

```
// produce a formatted string of output
int age = 42;
int iq = 95;
ostringstream output;
output << "Zoidberg's age is " << age << endl;
output << " and his IQ is " << iq << "!" << endl;
string result = output.str();

// result = "Zoidberg's age is 42\nand his IQ is 95!\n"
```

Exercise: inputStats2

- Write a function **`inputStats2`** that prints statistics about the data in a file. Example file, `carroll.txt` :

```
1  Beware the Jabberwock, my son,
2  the jaws that bite, the claws that catch,
3
4  Beware the JubJub bird and shun
5  the frumious bandersnatch.
```

- The call of `inputStats2("carroll.txt");` should print:

```
Line 1: 30 chars, 5 words
Line 2: 41 chars, 8 words
Line 3: 0 chars, 0 words
Line 4: 31 chars, 6 words
Line 5: 26 chars, 3 words
longest = 41, average = 25.6
```

inputStats2 solution

```
/* Prints length/count statistics about data in the given file. */
void inputStats2(string filename) {
    ifstream input;
    input.open(filename);

    int lineCount = 0, longest = 0, totalChars = 0;
    string line;
    while (getline(input, line)) {
        lineCount++;
        totalChars += line.length();
        longest = max(longest, line.length());
        int wordCount = countWords(line);   // on next slide
        cout << "Line " << lineCount << ": " << line.length()
             << " chars, " << wordCount << "words" << endl;
    }
    double average = (double) totalChars / lineCount;
    cout << longest = " << longest
         << ", average = " << average << endl;
}
```

inputStats2 solution

```
/* Returns the number of words in the given string. */
int countWords(string line) {
    istringstream words(line);
    int wordCount = 0;
    string word;
    while (words >> word) {
        wordCount++;
    }
    return wordCount;
}
```

File Output

- You can output to a file with `ofstream`.
  - Write to it using `<<`
  - Open and close it the same way as `ifstream`
  - Example:

```
ofstream outFile;
outFile.open ("output.txt");
outFile << "this is a test" << endl;
outFile.close();
```

  - The file will be created if it doesn't exist. If it does exist, it will be overwritten.

Formatted I/O

To use: `#include <iomanip>`

- helps produce formatted output, a la `printf`

Member name	Description
`setw(`***n***`)`	right-aligns next token in a field ***n*** chars wide
`setfill(`***ch***`)`	sets padding chars inserted by `setw` to the given char (default `' '`)
`setbase(`***b***`)`	prints future numeric tokens in base-***b***
`left, right`	left- or right-aligns tokens if `setw` is used
`setprecision(`***d***`)`	prints future doubles with ***d*** digits after decimal
`fixed`	prints future doubles with a fixed number of digits
`scientific`	prints future doubles in scientific notation

```
for (int i = 2; i <= 2000; i *= 10) {                  // 2        1.41
    cout << left  << setw(4) << i                      // 20       4.47
         << right << setw(8) << fixed                  // 200     14.14
         << setprecision(2) << sqrt(i) << endl;        // 2000    44.72
}
```