Introduction and basics of C++

CS 2124: Object Oriented Programming
Darryl Reeves, Ph.D.

Agenda

- Course Overview
- C++ Intro

Course Overview

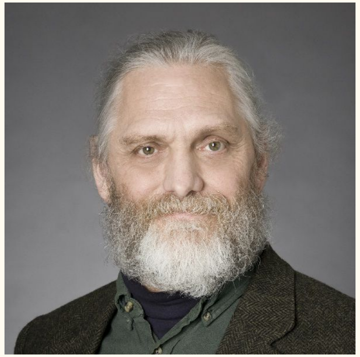
What is the Course About?

- Writing good, readable code
- Static typing
- OOP: Encapsulation, data hiding, delegation, inheritance, polymorphism…
- Addresses and pointers
- Memory management
- Operator Overloading
- Generic classes and functions
- Functors and lambda expressions
- Recursion! (Again? Yes!)
- STL

Instructors

- John Sterling
  - Office: 845, 370 Jay
  - john.sterling@nyu.edu

- Darryl Reeves
  - Office: 840, 370 Jay
  - dreeves@nyu.edu



Course grading

- Labs:10%
- Homework: 10%
- Midterms: 40% (20% each for two exams)
- Final exam: 40%
- Class participation!

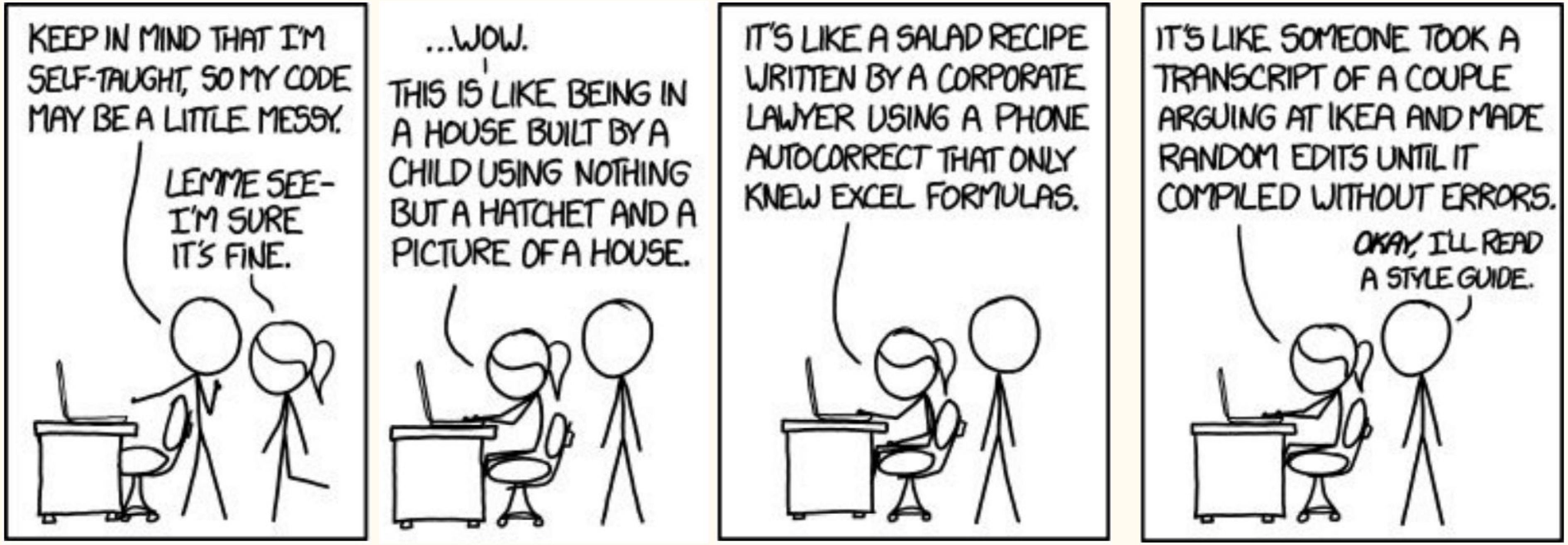
Labs / Recitations

- **Install C++ *before* the first lab!!!**
  - We use the terms “lab” and “recitation” interchangeably
- **<u>All</u>** lab solutions **must be submitted** on Brightspace
- Can be checked off in lab for full credit (still *must* submit!)
- If not checked off in lab, then will be graded when turned in… (scary!)
- Some labs are in the form of a “tutorial”
- Others are a single programming task
- Either way, it is expected that you *should* be able to complete it during lab.
- Labs are expected to be done *in lab*, not ahead of time, but are accepted till the end of the weekend (unless otherwise stated, e.g. rec14)
- Comments are not required in labs, but **good readable code** is.

Homework

- ~8 assignments for the semester
- Early ones will be typically be due in ~week.
- Later ones my be up to two weeks
- Late assignments (not labs) **are accepted** but with severe **penalty**.
- Code is to be <u>well-written</u>
  - **Well commented**
  - **Good naming** for variables, types and functions
  - **Good *use* of functions**
  - Avoid long functions
  - No long lines. **80 characters max / line**, even with comments
- Do <u>not</u> use features / libraries that have not yet been covered in class

Note on code readability


KEEP IN MIND THAT I'M SELF-TAUGHT, SO MY CODE MAY BE A LITTLE MESSY.
LEMME SEE— I'M SURE IT'S FINE.
...WOW.
THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.
IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.
IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS.
OKAY, I'LL READ A STYLE GUIDE.

Exams

- Consist of approximately 50% / 50% short and long questions
    - May be 60 / 40 or 40 / 60...
- Short answer questions
    - Most commonly, “what happens if we compile and run this code?”
        - Did it build / compile? Did it run without error? If so, what was its output?
        - Some students get confused by the difference between build errors and runtime errors
    - Some “short” answer questions may require a *few* lines of code.
    - No definitions, but if you don’t know the terminology, you may not understand the questions.
        - This is one reason why we correct your choice of words when you ask / answer in class.
- Long answer questions
    - Programming problems similar to (but shorter than) lab or homework questions
    - Comments are *not* required
    - Nor are “include” or “using” statements. (you’ll know what those are shortly)
    - Write clearly. If we can’t read it, ...
- BTW, do not provide two answers for one question.

Class participation

- TurningPoint used for active learning
  - questions will be interspersed throughout lectures
  - responses can be entered via computer/cell phone/tablet/etc
  - contact me if having trouble registering
- *Consistent* and *constructive* class participation *may* result in a grade boost
  - e.g. from B+ to A-
  - boost will depend on distance from next grade cut-off
  - random responses to questions not *constructive*
  - no penalty for not participating in TurningPoint activities

Questions?

- *Please* ask questions in class!
- If you have a question, likely others have the same one.
- We try to keep the lecture size down so that all students should be able to ask questions.
- Feel free to come to the office to ask questions!
  - Don't wait till it is too late for us to be able to help
- TAs will have office hours, too.
- Some students want to have their homework "looked at" *before* it is graded.
  - No, we won't do that.
  - We will give guidance on debugging or answer questions about what needs to be done
  - We just won't tell you if your code is good or bad or what sort of grade to expect. You should know.

Course information (where to find it)

- NYU Brightspace
  - Homework assignments
  - Labs
  - Grades
    - Goal is to have all submissions graded within a week
  - Lecture slides
  - Discussion forum
  - In-class code
- Professor Sterling's lecture notes: cis.poly.edu/jsterling/cs2124/Notes/Syllabus.html
  - Note that there is no course textbook
- Good C++ language reference: http://www.cplusplus.com/

Development environment

- Install a version of C++ on your machine *before* coming to lab!
- We don't force you to use one environment or another
- We recommend:
    - On Windows: https://visualstudio.microsoft.com/
    - On Mac: https://developer.apple.com/xcode/
    - Linux: g++. I expect you already have it.
    - CLion
- A simple setup includes
    - good text editor
    - command line compiler (g++/clang)

C++ Introduction

C++ history

- Created by **Bjarne Stroustrup**
    - Danish Computer Scientist
    - Bell Labs (1979)
    - Inspired by Simula (OOP) and C (speed/portability)
    - http://www.stroustrup.com/

C++ history

- Created by **Bjarne Stroustrup**
  - Danish Computer Scientist
  - Bell Labs (1979)
  - Inspired by Simula (OOP) and C (speed/portability)
  - http://www.stroustrup.com/
- Continues to evolve
  - Current standard: `C++20`
  - modern `C++`: a completely type-safe and resource-safe language

The simplest C++ program

```
int main() {
   return 0;
}
```

- Every program *must* have a function named `main`
- Blocks of code are surrounded by *braces*: `{}`
- Statements end with a semicolon: `;`
- Each function must state the type it returns
  - type returned by `main` is `int` (an integer)
- `main` function *must* return an integer
  - `0` means successful completion
- Every function that returns a value *must* include a `return` statement

The simpler, simplest C++ program

```
int main() {
   // return 0;
}
```

(other than `main`)

- Every function ^ that returns a value *must* include a return statement
- Without a `return` statement, `main` returns `0` (by default)
- Comments begin with `//`

Hello CS2124!

```
#include <iostream>

int main() {
   std::cout << "Hello CS2124!\n";
}
```

- `<<` is the output operator
    - target *stream* appears on the left
    - what is output on the right
    - angle brackets "point to" the output stream
- `std::cout` represents standard output (the screen, by default)
- string *literals* are always surrounded by double quotes: "   "
- `#include` specifies a library for compiler
- `iostream` provides the definition for `std::cout` (and much more)

Hello CS2124!

```
#include <iostream>
using namespace std;

int main() {
    std::cout << "Hello CS2124!\n";

}
```

Hello CS2124!

```
#include <iostream>
using namespace std;

int main() {
    // std::cout << "Hello CS2124!\n";
    cout << "Hello CS2124!\n";
}
```

- `using namespace std;`
  - allows reference to `cout` and other symbols without needing to type `std::`

Hello CS2124!

```
#include <iostream>
using namespace std;

int main() {
    // std::cout << "Hello CS2124!\n";
    cout << "Hello CS2124!" << endl;
}
```

- output can be chained
- `endl`: an end-of-line character
  - full reference: `std::endl`
  - alternative to writing `"\n"`
  - also flushes the output stream (not crucial to understand now)

One key difference between C++ and Python

- variables are declared to have a **type**
  - only things of that type can be stored in the variable
  - examples:
    - `int num;`
    - `double score;`
    - `string name;`
- specifying types also done for
  - function parameters
  - function return types
- other programming languages work similarly (e.g. C, C#, Java)

Variables

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    int age = 42;
    double pi = 3.14159;
    string txt = "the cat in the hat";
    cout << "age: " << age << ", pi: " << pi << ", txt: " << txt << endl;
}
```

- `#include` for `string` required because strings are not primitive / built-in **type**

An uninitialized variable

```
int main() {
   int age;
   cout << "age: " << age << endl;
}
```

leaving out #include/using statements to save space on slide

- behavior undefined when accessing value of uninitialized variable

```
age: 0
```

- no guarantees of value of uninitialized variable
- some compilers warn of uninitialized variables
- **always** provide an initial value when declaring variable
- strings behave differently -- automatically initialized to empty string

An uninitialized variable (cont.)

```
void foo() {
    int num1 = 17;
    cout << "num1: " << num1 << endl;
}
void bar() {
    int num2;
    // Not initialized!!!
    cout << "num2: " << num2 << endl;
}
int main() {
    foo();
    bar();
}
```

```
num1: 17
num2: 17
```

- `num2` using same memory location as `num1`
- again, behavior undefined

Getting input

```
int main() {
    int i_var = -1;
    cout << "i_var: " << i_var << endl;
    cout << "input an integer value: ";
    cin >> i_var;
    cout << "i_var: " << i_var << endl;
}
```

```
i_var: -1
input an integer value: 72
i_var: 72
```

- **cin** is *standard input* (the keyboard by default)
- angle brackets "point" from stream to variable

Conditions

```
int main() {
    int a_num;
    cout << "a_num? ";
    cin >> a_num;
    if (a_num == 6) {
        cout << "a_num is a small perfect number" << endl;
    } else if (a_num == 42) {
        cout << "a_num is the answer" << endl;
    } else {
        cout << "a_num is something else" << endl;
    }
}
```

- `if`, `else if`, and `else`
- condition defined inside parenthesis
- code to handle condition located in block (surrounded by `{}`)

Logical operations

```
int main() {

      int a_num;
      cout << "a_num? ";
      cin >> a_num;

      if (a_num == 6 || a_num == 28) {
           cout << "a_num is a small perfect number\n";
      } else if (a_num >= 0 && a_num <= 9) {
            // Note: 0 <= a_num < 10 not possible
            cout << "a_num is an imperfect single digit number\n";
      } else if (!(a_num < 0 || a_num > 99)) {
           cout << "a_num is a two digit number\n";
      } else {
          cout << "a_num is something else\n";
      }
}
```

- and: `&&`
- or: `||`
- not: `!`