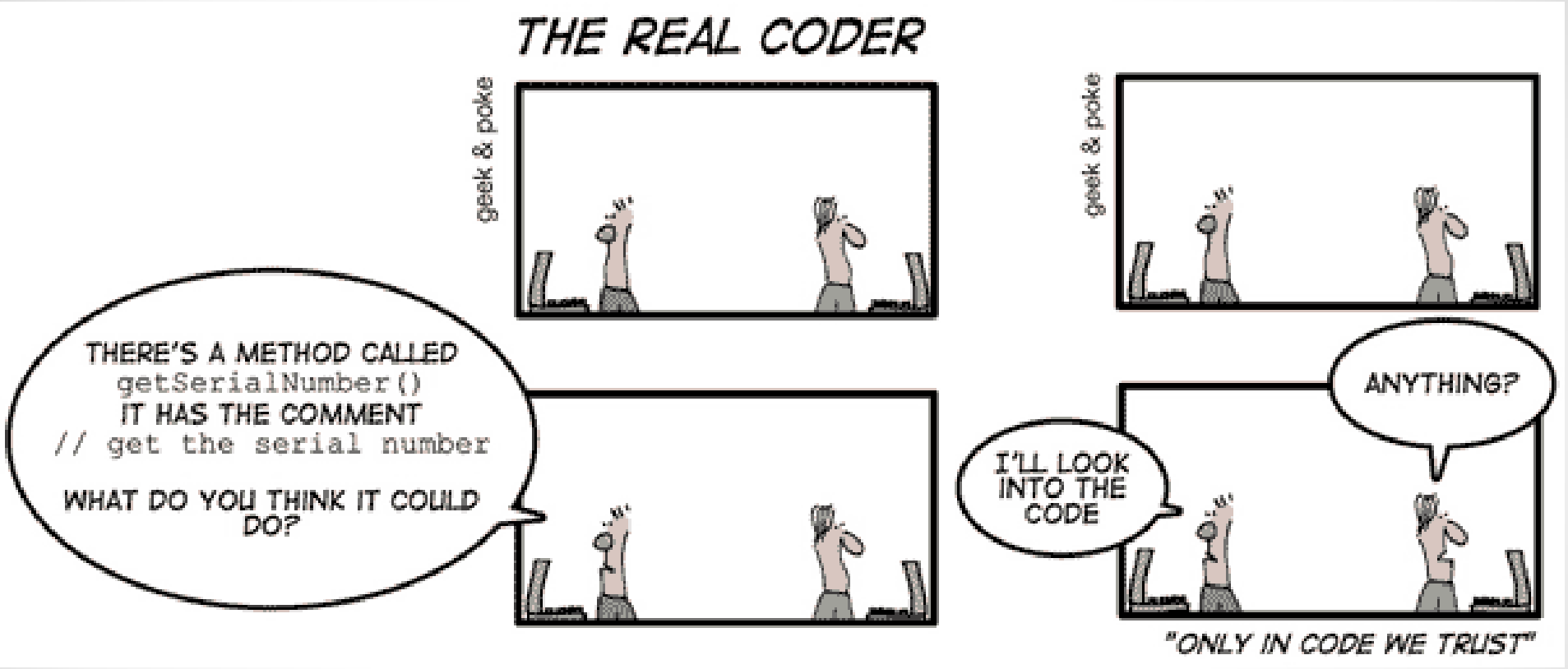
C++

SWE315

Lesson 3

Prof. Zachi Baharav
zbaharav@cogswell.edu


THE REAL CODER
geek & poke
THERE'S A METHOD CALLED
getSerialNumber()
IT HAS THE COMMENT
// get the serial number
WHAT DO YOU THINK IT COULD DO?
geek & poke
I'LL LOOK INTO THE CODE
ANYTHING?
"ONLY IN CODE WE TRUST"

Lesson 3

- In this lesson:
  - **Object Oriented Programming !!**
    - Our first class
  - Compound variables
    - Array
    - Enumerator
    - We will mostly skip for now:
      - Structures
      - Unions
  - Functions:
    - Default parameters
  - Programs:
    - Dungeon Crawl
    - Multiple files
  - Homework 3
  - Lab work

Arrays

- typeName arrayName [arraySize] ;
- Example

  int months[12];
- 12 elements: Starts at 0, ends at 11.
- Initialize:
  - int cards[4] = {3, 6, 8, 10}; //OK
  - float hotelTips[5] = {5.0, 2.5}; // OK. The rest are initialized to zero
  - int total[500] = {0}; // The first is set to zero, and all the rest as well.

Enumeration

enum spectrum {red, orange, yellow, blue};

- This statement does two things:
  - spectrum is the name of a new type! Spectrum is termed enumeration.
  - Established red, orange, yellow and so on as symbolic constants for 0-3. These are enumerators.
- Valid use:

```
spectrum band;
band = blue;
```

- Setting values:

```
enum bits {one=1, two=2, four =4, eight=8};
```

- **Tricks are available:** Converting enums to integers ; setting multiple with same values, range, etc.

Default arguments

- Value that is used automatically if you omit the corresponding actual argument.
  - Defined in the function prototype! (so the compiler knows)
    - And ONLY in the prototype. Function definition is the same.
  - Defaults are 'from right to left'. Namely, all the ones beyond one have default values

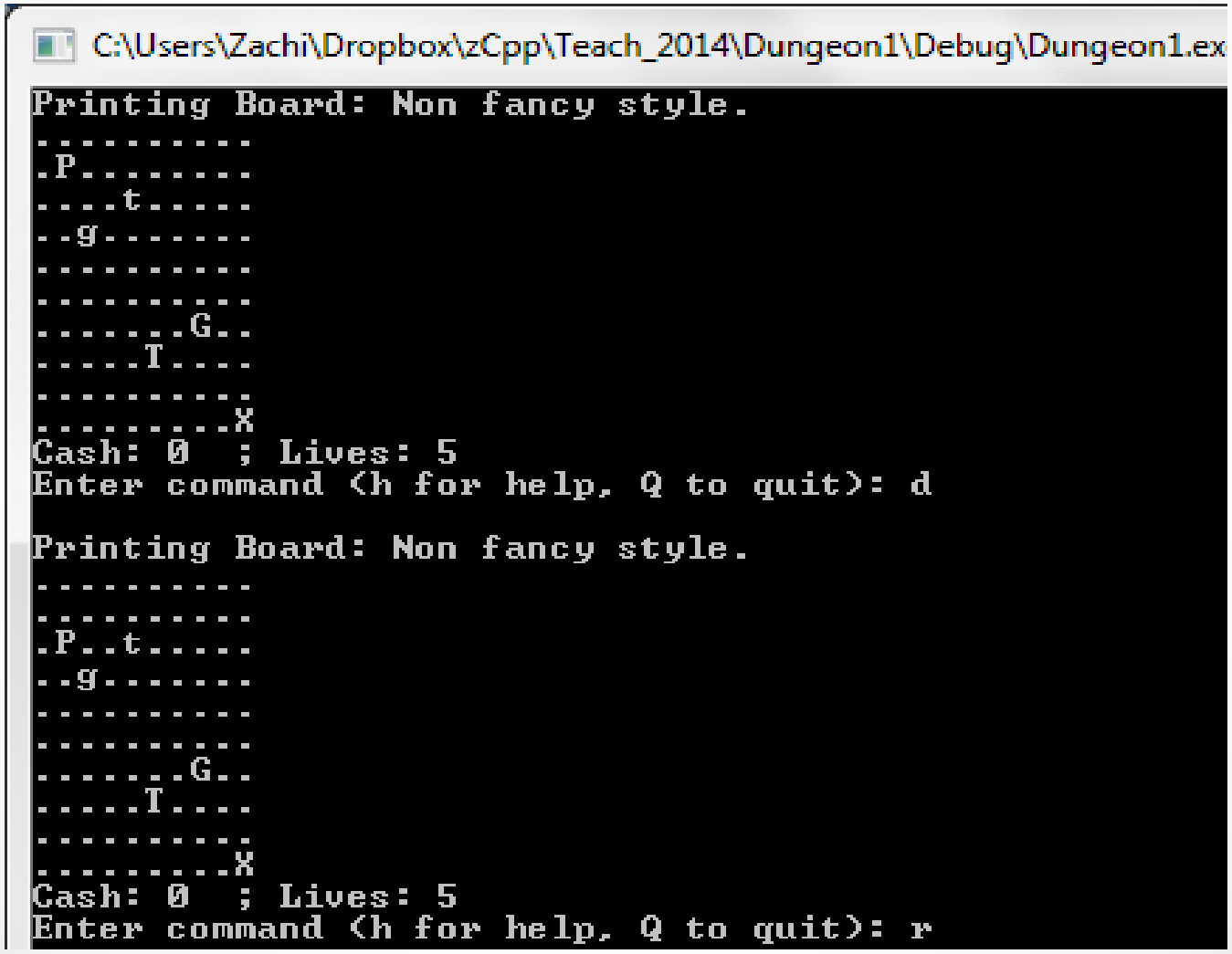
  - Example:

```
int harpo(int n, int m=4, int  j=5); // VALID
int harpo(int n, int m=4, int  j);     // INVALID


beeps  = harpo(2);     // VALID
beeps  = harpo(1,8);  // VALID
beeps  = harpo(8,7,6);              // VALID
```

- **Default arguments are NOT a major programming breakthrough**
  - They are convenience
  - Reduce the number of constructors/methods/methods-overloads you have to define.

Dungeon Crawl



OOP

- Procedural and Object-Oriented Programming
  - Procedural: What's needed from the program, how to break it down to functions and segments, program structure.
    - Write a program.
  - OOP: What are the objects handled, what needed from them.
    - Write objects.
    - Write the calling program.

- With OOP approach you concentrate on the object:
  - How does the user perceives it?
  - What data is described?
  - What operations are needed?

Class

- Translating abstraction to a user-defined type.
  - Abstraction : Hiding all the details, but simply looking at the overall interface with the user.
- Enables easily:
  - **Abstraction**
  - **Encapsulation and Data-hiding** – Methods and data.
  - **Polymorphism** - Same interface to apply to different types. (E.g., operator overloading, function overloading, etc).
  - Inheritance
  - Reusability of code
- Combines:
  - Data representation: Class declaration.
  - Data manipulation : Methods.

END